MODEL PREDICTIVE CONTROL OF WHEELED MOBILE ROBOTS

By

Haris Chowdhry

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Masters of Applied Science
in
The Faculty of Electrical and Computer Engineering
Program
University of Ontario Institute of Technology
December 2010

CERTIFICATE OF APPROVAL

Submitted by **Haris Chowdhry**

In partial fulfillment of the requirements for the degree of

Master of Applied Science in **Electrical and Computer Engineering**

Date of Defence: **December 2, 2010**

Thesis title: Model Predictive Control of Wheeled Mobile Robots

Examining Committee

Hossam Kishawy
Chair of Examining Committee

Andrew Hogue
External Examiner

Ruth Milman
Research Supervisor

N/A
Research Co-Supervisor

Shahryar Rahnamayan
Examining Committee Member

Lixuan Lu
Examining Committee Member

As research supervisor for the above student, I certify that I have read and approved changes required by the final examiners and recommend the thesis for acceptance:

Ruth Milman
Research Supervisor

Abstract

The control of nonholonomic wheeled mobile robots (WMRs) has gained a lot of attention in the field of robotics over the past few decades as WMRs provide an increased range of motion resulting in a larger workspace. This research focuses on the application of Model Predictive Control (MPC) for real-time trajectory tracking of a nonholonomic WMR. MPC is a control strategy in which the control law is designed based on optimizing a cost function. The input and output constraints that may arise in practical situations can be directly incorporated into the control system using MPC. Computation time is the biggest hurdle in adapting MPC strategies for trajectory tracking. This research applies a non-feasible active set MPC algorithm developed in [1] which is faster than the traditional active set methods (ASMs). A discrete-time linear model of a general WMR is used for the simulation. MATLAB simulations are performed for tracking circular as well as square trajectories using the discretized WMR model and the non-feasible ASM (NF-ASM). The performance of NF-ASM is compared to two other well-known traditional algorithms, i.e. Fletcher's ASM and MATLAB's Quadratic Programming algorithm. It is shown that, although all these algorithms are capable of providing satisfactory trajectory tracking performance, NF-ASM is a better choice in terms of the simulation time and required number of iterations for real-time trajectory tracking of any type as long as the constraints on the inputs stay active for a long period during the simulation.

Keywords: MPC, Mobile Robots, Non-Holonomic, Active Set, Non-Feasible.

To my Father and Maa Ji

Acknowledgements

I would like to thank my supervisor, Dr. Ruth Milman, for setting up a stress free atmosphere for my learning and for being so helpful and kind. I really gained a lot from her; not just work-related knowledge but also, personal values like hard work, honesty and dedication that will stay with me for the rest of my life.

I am also thankful to all my professors at UOIT who taught me advanced techniques in control design and optimization. I would particularly like to acknowledge the help of Dr. Shahryar Rahnamayan in advanced optimization, Dr. Hossam Gabbar in CANDU reactor control and Dr. Dan Zhang in robotics and automation techniques. The invaluable experience gained under their supervision would definitely be beneficial in the coming years.

Of course, I am grateful to my parents and grandparents for their constant support and *love*. Without their prayers, this work would never have come into existence.

Oshawa, Ontario — Haris Chowdhry
November 10, 2010

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction: Wheeled Mobile Robot Control

Mobile robot control is an area of research that has attracted a lot of attention recently due to the need for autonomous systems. Although the field of robotics has achieved great success in the manufacturing sector, these industrial robots suffer from the major disadvantage of the lack of mobility. Robot manipulators with a fixed base allow limited range of motion and a significantly small workspace compared to robots mounted on movable bases, i.e. a Wheeled Mobile Robots (WMRs).

When applying control to a standard moving vehicle setup, it is important to realize that the motion of the wheels is restricted thus only some instantaneous directions of motion can be achieved. As an example, consider the parallel parking manoeuvre for the car shown in Fig. 1.1. It is not possible to drive a car sideways, as shown using the dotted arrow line in the figure, to achieve parallel parking. One has to make a $ritualistic$ manoeuvre (shown with solid lines in figure) of first moving parallel to the car upfront and then reversing with the wheels turned towards the curb to achieve parallel parking. This type of system, where it is not possible to achieve motion in certain directions in an instantaneous manner, is called a nonholonomic system.

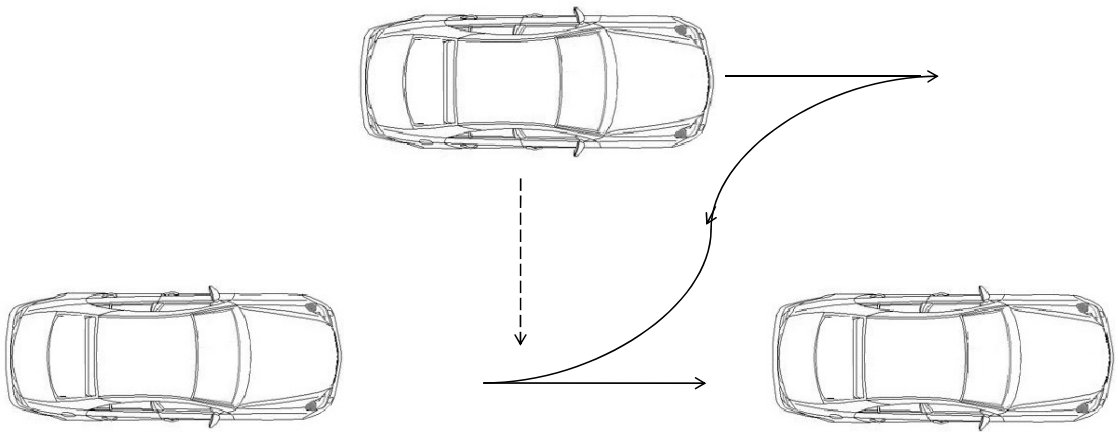

Figure 1.1: Parallel Parking of a WMR

There is extensive literature available on the introduction and problem formulation for nonholonomic systems. The book by Sastry, Murray and Li [2] offers a good introduction to the nonholonomic systems. Another book by Neimark and Fufaev [3] explains the dynamic modeling of nonholonomic systems.

The control design theory of simple WMRs is developed in [4, 5, 6] and example applications of variants of this theory are presented in [7, 8, 9]. Optimal control design techniques and their applications to nonholonomic WMRs are detailed in [10, 11]. The path planning of WMRs in the presence of obstacles is explained in [12, 13] and in the presence of constraints is given in [14, 15]. Recently, adaptive techniques are also used to control WMRs [16, 17]. The control design theory of WMRs with trailers attached to them is developed in [18, 19] and example applications of variants of this theory are presented in [20, 21, 22].

1.1 Model Classification

The control modeling of a nonholonomic system is generally classified into two groups [23]:

1) The kinematic model

2) The dynamic model

The kinematic model provides the mathematics of motion of a WMR without considering the forces responsible for that motion. On the other hand, the dynamic model takes into account the forces causing the WMR to move.

1.1.1 Kinematic Model

In a kinematic model, the control inputs to the system are generally linear and angular velocity variables. The kinematic model of a nonholonomic system is given by the following equation [23]:

$$\dot{x} = g_1(x)u_1 + g_2(x)u_2 + ... + g_m(x)u_m \tag{1.1.1}$$

where $x = [x_1 x_2 ... x_n]'$ is the state vector, $g_i(x)$ and u_i $(i = 1, 2, ...m)$ are the specified vector fields and the control inputs , respectively.

In [2], it is shown that the kinematic model of a system is completely nonholonomic if the rank of the controllability Lie algebra computed by the iterated Lie brackets of $g_1(x), g_2(x), ..., g_m(x)$ is n. Lie algebra makes it possible to determine the controllability of nonlinear systems. The complete nonholonomy of a system implies that all points in the space are reachable even under the imposition of constraints. Hence, the kinematic model of the system is completely controllable as long as it is completely nonholonomic.

For simplicity of the control design, the model described in (1) is transformed into different forms. For example, for a system with two inputs, i.e. $m = 2$, the chained form in which the derivative of each successive state is dependent on the previous state [23]:

$$\dot{z_1} = y_1 \dot{y_2}$$

$$\dot{z_2} = z_1 \dot{y_2}$$

$$\dot{z_3} = z_2 \dot{y_2}$$

$$\vdots$$

$$\dot{z}_{n-m} = z_{n-m-1} \dot{y_2}$$

$$\dot{y_1} = u_1$$

$$\dot{y_2} = u_2$$

and the power form in which the derivative of each successive state is dependent on an increasing power of the derivative of the control signal [23]:

$$\dot{z_1} = y_1 \dot{y_2}$$

$$\dot{z_2} = \frac{1}{2}(y_1)^2 \dot{y_2}$$

$$\vdots$$

$$\dot{z}_{n-m} = \frac{1}{(n-m)!}(y_1)^{n-m} \dot{y_2}$$

$$\dot{y_1} = u_1$$

$$\dot{y_2} = u_2$$

It is important to note that the model in (1.1.1) not only includes the evolutionary dynamics of the system but it also incorporates the constraints. Hence, the control of this type of system will automatically abide by the constraints imposed on the system.

A traditional example of a kinematic model is the Reed-Shepp's car [24]:

$$\begin{aligned} \dot{x_c} &= v\cos\theta \\ \dot{y_c} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{1.1.2}$$

where (x_c, y_c) is the geometric center as well as the center of the mass of the WMR, θ denotes the orientation of the WMR with respect to the horizontal axis, v is the linear velocity and ω is the angular velocity of the wheels. Since the wheels cannot move sideways, we have the following nonholonomic constraint that must be imposed on the system:

$$\dot{x_c}\sin\theta - \dot{y_c}\cos\theta = 0$$

The control inputs to (1.1.2) are the forward velocity v and the angular velocity ω of the WMR. By using the following state and control transformation:

$$\begin{aligned} x_1 &= x_c\cos\theta + y_c\sin\theta \\ x_2 &= \theta \\ x_3 &= x_c\sin\theta - y_c\cos\theta \\ u_1 &= v - \omega x_3 \\ u_2 &= \omega \end{aligned}$$

we obtain the kinematic power form:

$$\dot{x_1} = u_1$$

$$\dot{x_2} = u_2$$

$$\dot{x_3} = x_1\dot{x_2}$$

In fact, any system with three states and two inputs can be converted to the kinematic power form [105].

1.1.2 Dynamic Model

In a dynamic model, the control inputs to the system are generally force and torque variables. The dynamic model of a nonholonomic system is given by the following equation [23]:

$$q_i{}^{r_i} = u_i, i = 1, 2, ..., m \tag{1.1.3}$$

subject to constraints of the form:

$$\dot{x} = g_1(x)q_1 + g_2(x)q_2 + ... + g_m(x)q_m \tag{1.1.4}$$

where $x = [x_1 x_2 ... x_n]'$ is the state vector, q_i and r_i $(i = 1, 2, ...m)$ are the outputs and the order of the time differentiations, respectively.

It is assumed that the constraints described in (1.1.4) are completely nonholonomic. Just as in the case of the kinematic model, the dynamic model described by (1.1.3) and (1.1.4) can be transformed into a single set of equations in chained or power form. As an example, consider again the model of the WMR described in (1.1.2). We can obtain the following dynamic model for the system [23]:

$$\ddot{y} = \frac{\lambda}{m}\sin\theta + \frac{F_1}{m}\cos\theta$$
$$\ddot{y_c} = -\frac{\lambda}{m}\cos\theta + \frac{F_1}{m}\sin\theta$$
$$\ddot{\theta} = \frac{\tau}{I_c}$$
$$\ddot{x_c}\sin\theta - \ddot{y_c}\cos\theta = 0 \tag{1.1.5}$$

where F_1 is the pushing force in the direction of the heading angle, τ_1 is the steering torque about the vertical axis through the center of the mass, m is the mass of the robot, I_c is the moment of inertia and λ is the scalar constraint multiplier.

The following power form representation of the WMR can easily be obtained from (1.1.5):

$$\ddot{x_1} = u_1$$

$$\ddot{x_2} = u_2$$

$$\ddot{x_3} = x_1\dot{x_2}$$

The details of this transformation can be found in [25].

1.2 The WMR Model

A WMR consists of a solid base with an optionally mounted manipulator arm which is used to perform a predefined task. The base has wheels to add mobility. Since this research focuses on the motion tracking of the WMR, it is assumed that the WMR is not equipped with a manipulator. This leads to a simple schematic of the WMR given in Fig. 1.2. Also,

this research focuses on the impact of changing the speed and steering velocity on the motion of the WMR. Therefore, kinematic modeling of the WMR is more suited as compared to dynamic modeling.

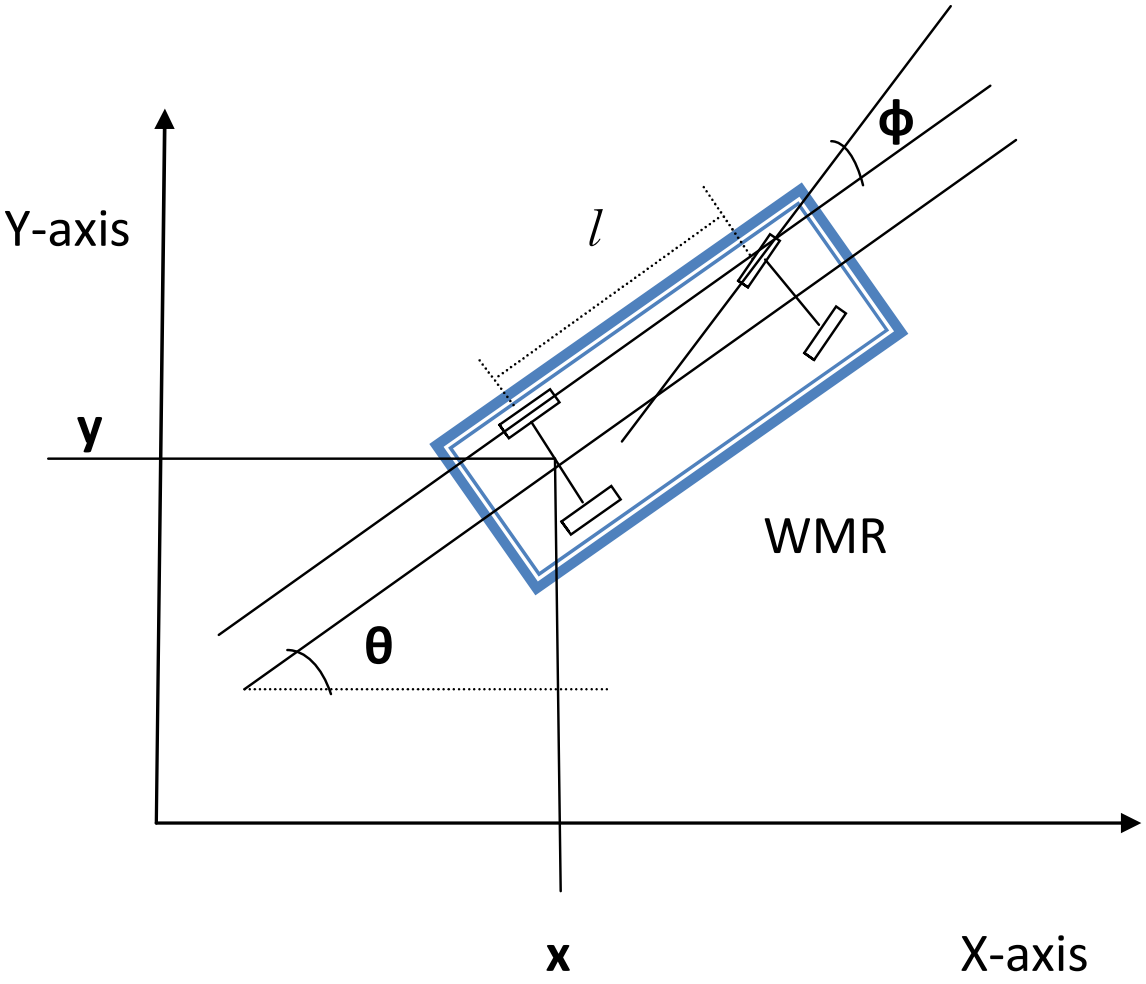

Figure 1.2: The Orientation and Position of the WMR

In order to find a kinematic model for the WMR, it is assumed that the robot is a rigid body and has non-deforming wheels. It is also assumed that the robot moves without slipping, i.e. the translational motion of the robot is solely due to the rolling of the wheels on the ground. Under these assumptions, the nonlinear rear-wheeled kinematic model of

the WMR, shown in Fig. 1.2, is given by [26]:

$$\dot{x}(t) = v_1(t)\cos\theta(t)$$

$$\dot{y}(t) = v_1(t)\sin\theta(t)$$

$$\dot{\theta}(t) = v_1(t)\frac{\tan\phi(t)}{l}$$

$$\dot{\phi}(t) = v_2(t) \tag{1.2.1}$$

where (x, y) represents the position of the robot, θ denotes the orientation of the WMR with respect to the horizontal axis, ϕ is the steering angle and l is the distance between the two axles of the car as shown in Fig. 1.2. The control inputs to (1.2.1) are the linear velocity of the WMR, v_1, and its steering velocity, v_2. The $\begin{bmatrix} x & y & \theta & \phi \end{bmatrix}$ in the above representation completely defines the location and orientation of a WMR at a particular instance in time.

The Model given in (1.2.1) is nonlinear. To simplify the model for trajectory tracking, a linearized error model is developed by computing the Jacobian matrix of the nonlinear system defined in (1.2.1) and evaluating it at the reference point $(x_d, y_d, \theta_d, \phi_d)$. This linearized state space model of the WMR is given below [27]:

$$\dot{\tilde{X}}(t) = A(t)\tilde{X}(t) + B(t)\tilde{u}(t) \tag{1.2.2}$$

where

$$\tilde{X}(t) \doteq X(t) - X_d(t) = \begin{bmatrix} x(t) - x_d(t) \\ y(t) - y_d(t) \\ \theta(t) - \theta_d(t) \\ \phi(t) - \phi_d(t) \end{bmatrix}$$

$$\tilde{u}(t) \doteq u(t) - u_d(t) = \begin{bmatrix} v_1(t) - v_{d1}(t) \\ v_2(t) - v_{d2}(t) \end{bmatrix}$$

$$A(t) \doteq \begin{bmatrix} 0 & 0 & -v_{d1}(t)\sin\theta_d(t) & 0 \\ 0 & 0 & v_{d1}(t)\cos\theta_d(t) & 0 \\ 0 & 0 & 0 & \frac{v_{d1}(t)}{lcos^2\theta_d(t)} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B(t) \doteq \begin{bmatrix} \cos\theta_d(t) & 0 \\ \sin\theta_d(t) & 0 \\ \frac{\tan\theta_d(t)}{l} & 0 \\ 0 & 1 \end{bmatrix}$$

In the above linearized model (1.2.2), the two control inputs are the difference in the actual and desired linear velocities $(v_1 - v_{d1})$ and the difference in the actual and desired steering velocities $(v_2 - v_{d2})$. In this research, we put constraints on these control inputs. In other words, we put constraints on the deviation of the linear and steering velocities from their reference values.

1.3 Controllability of the WMR

Controllability of a linear system is defined as its ability to achieve any point in its state-space by using bounded control inputs [26]. To determine whether a linear system is controllable, a controllability matrix is determined from the state space form of that system. A linear system is said to be controllable if its controllability matrix has full rank. For a linear system of the form (1.2.2) with system matrix $A(t)$ of size $n \times n$, the controllability matrix is given by:

$$Ctrb \doteq \begin{bmatrix} B(t) & A(t)B(t) & A^2(t)B(t) & \ldots & A^{n-1}B(t) \end{bmatrix}$$

If the rank of $Ctrb$ is n, the linear system is controllable. In other words, it is possible to achieve any point in the state-space of the system by using bounded inputs.

By computing the controllability matrix of the linearized WMR model developed in the previous section, we get:

$$Ctrb = \begin{bmatrix} \cos\theta_d(t) & 0 & \frac{-v_{d1}(t)\sin\theta_d(t)\tan\theta_d(t)}{l} & 0 & 0 & \frac{-v_{d1}^2(t)\sin\theta_d(t)}{l\cos^2\theta_d(t)} & 0 & 0 \\ \sin\theta_d(t) & 0 & \frac{v_{d1}(t)\cos\theta_d(t)\tan\theta_d(t)}{l} & 0 & 0 & \frac{v_{d1}^2(t)\cos\theta_d(t)}{l\cos^2\theta_d(t)} & 0 & 0 \\ \frac{\tan\theta_d(t)}{l} & 0 & 0 & \frac{v_{d1}(t)}{l\cos^2\theta_d(t)} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Since all the rows and the first four columns of $Ctrb$ are independent of each other as long as $v_d1(t)$ is not zero, the rank of $Ctrb$ is 4 which is equal to the size of the system matrix $A(t)$. When $v_{d1}(t)$ becomes zero, the above controllability matrix loses full rank. Hence, the linearized WMR model in (1.2.2) is completely controllable as long as the reference input velocity $v_{d1}(t)$ is not zero. This result allows us to use MPC for trajectory tracking [26]. The continuous-time state-space model of (1.2.2) is converted into discrete-time model and is used for optimization.

1.4 Problem Statement

The purpose of this research is to establish a trajectory tracking control technique for non-holonomic WMRs. While doing so, we would like to ensure that the system is able to:

1. Perform real-time control of the WMR.
2. Allow for an easy application of input and state constraints that may arise in practical situations.
3. Allow for the tuning of the final control law to obtain good transient performance.

4. Allow for the control design optimization in a systematic manner.

Although different control techniques have been established over the years to solve the WMR trajectory tracking problem [4, 5, 6], most of them do not allow easy control law tuning and incorporation of constraints. Model Predictive Control (MPC) technique meets the design requirements $2-4$. However, traditionally, it is not used for the control of WMR due to its high computational burden and high time consumption. In this research, a linear MPC is applied which uses a non-feasible active set method (NF-ASM) from [1]. This ASM is much faster than the traditional active set methods. To compare the performance of this ASM, Fletcher's Active Set Method (FL-ASM) developed in [28] and MATLAB's Quadratic Programming (QP) algorithm are also applied to the WMR.

1.5 Structure of the Thesis

As explained in the previous section, this research uses NF-ASM with the linear MPC technique for the trajectory tracking of a WMR. To compare the performance of NF-ASM, two traditional methods, namely FL-ASM and MATLAB's Quadratic Programming (QP) algorithm, are also used to control the WMR and make it track different types of trajectories. NF-ASM uses a different approach from the traditional ASMs in that the intermediate solutions of the algorithm do not enforce all constraints and hence, multiple active set changes can be made per iteration. Experimental results have shown that the applied algorithm can converge faster than other algorithms by as much as 40 times.

In Chapter 2, a review of control literature on nonholonomic systems is conducted. The MPC algorithms used for trajectory tracking and cost function minimization are explained in Chapter 3. The overall control structure and simulation setup is presented in Chapter

4. The simulation results are discussed in Chapter 5 and a brief conclusion of the thesis is presented in Chapter 6. Finally, the MATLAB code is presented in the appendices.

Chapter 2

Control of Nonholonomic Systems

It is not straightforward to control a nonholonomic system, as opposed to a holonomic system where the motion is possible in any direction without restrictions. The relative difficulty of the control problem depends on the nature of the nonholonomic system (i.e. its model) as well as the control objective. It is possible to use classical nonlinear control techniques if the control objectives are to stabilize a system to certain trajectories [29], stabilization around a region where the equilibrium exists [30, 31, 32], dynamic path following [33, 34] and output tracking [35, 36]. On the other hand, the standard nonlinear control techniques cannot be used for control objectives such as motion planning [2, 37, 38] and stabilization to an equilibrium state.

2.1 Traditional Nonholonomic Control Techniques

In [39] and [40], authors show that it is neither possible to stabilize the nonlinear plant directly with time invariant feedback nor possible to apply nonholonomic integrators or schemes such as feedback linearization.

Over the years, many techniques have been employed to mitigate the aforementioned

problem. Sliding mode control is employed in [41]. Time-varying feedback laws [42], hybrid control laws [43] and dynamic feedback linearization [44] are also used to solve the above problem. Recently, robust adaptive controls [17, 16] and MPC are also used to control a WMR. The next sections provide a brief overview of all these techniques and the advantages/disadvantages associated with them.

2.1.1 Sliding Mode Control

The sliding mode technique develops different time-invariant control laws that ensure the Lyapunov stability of the nonholonomic system. The state feedback control law switches between these time-invariant laws and is, therefore, not continuous in time. The switching decisions are based on the type of tracked trajectory. The sliding mode approach transforms a higher-order system into first-order system which simplifies control design and analysis.

In [41], a control law for the stabilization and tracking problem of a nonholonomic system is developed. By utilizing sliding mode theory, a feedback law is generated that globally asymptotically stabilizes WMR to origin.

The major disadvantage in the development of this technique is that the initial conditions of the system need to be specified and are bounded within a region. This limits the scope of this technique in practical situations. Also, the overall feedback law is discontinuous. This discontinuity could cause erroneous results if the delays in the system are not considered carefully. The major advantages of this technique are robustness and high accuracy. However, this technique is sensitive to parameter variation and the sudden switching between the controllers causes high frequency variation which creates ”chattering effects”. The switching delay, which is defined as the time interval between the deviation from the constraints to the actual measurement time, also affects the performance of the system. The

increase in switching delay raises the probability of a constraint violation as the controller is not able to react immediately to any deviation in the system. It is possible to reduce the response time of the system by using a very high feedback gain but this method increases the required control effort and hence, is seldom used in practical scenarios. Another disadvantage of this technique is that the control design optimization is not intuitively possible.

2.1.2 Time-varying Feedback Laws and Dynamic Feedback Linearization

In [44], separate control techniques are applied for trajectory tracking and posture stabilization. Three different techniques are used for trajectory tracking and their results are compared: 1) Dynamic feedback linearization using a nonlinear controller, 2) Simple linear feedback design of an input to linearize the error dynamic around a reference trajectory and 3) Nonlinear feedback design of the input based on Lyapunov function. It is shown that the dynamic feedback linearization control performs better than other controllers in terms of tracking error. It was observed that for a simple linear feedback design, the initial conditions also affect that trajectory tracking as larger tracking error is observed when a non-zero input velocity is used. The main disadvantage of the above controller implementations is that there is no easy way to incorporate output constraints. Also, only a smooth eight-shaped path is tracked. Abruptly changing trajectories such as a square or a rectangular path is not tracked. It is expected that the tracking error at the corners of such trajectories would be sufficiently high since PD gains are used to tune the controller. Increasing the PD gains would be one way to reduce the tracking error but it would slow down the speed of the controller. Since WMR requires real-time control, computational workload is one important aspect to be used for controller performance comparison. However, no time

requirements to perform the trajectory tracking task are given in the paper.

For posture stabilization, four different controllers are used in [44]: 1) Smooth time varying stabilization 2) Non-smooth time varying stabilization 3) Design using polar coordinates and 4) Dynamic feedback linearization. It is shown that smooth and non-smooth time varying stabilizations are very slow, erratic and hard to tune. On the other hand, design with polar coordinates and dynamic feedback linearization are fast and the response is natural as opposed to erratic in the case of other controllers. It is also easy to tune and optimize the final controller using these two controllers especially in case of dynamic feedback linearization since PD control gains are used.

The major advantage of dynamic feedback linearization over polar coordinates is that the designed controller is in a general form and can be applied to different WMR models. The major disadvantage of the WMR controller design explained in [44] is that separate controllers are used for posture stabilization and trajectory tracking. Hence, the overall control effort is significant. Also, robustness in terms of disturbance rejection is not tested.

2.1.3 Hybrid Control Laws

Hybrid control laws make use of continuous time as well as discrete time controllers to achieve feedback stabilization. A two-level control structure is used in hybrid control strategy, in which the top-level controller is a discrete-time switching control which switches between multiple continuous time or discrete time low-level controllers based on a predefined set of rules. This control technique is applicable to both chained form [45, 46, 47] and power form nonholonomic systems [48]. The chained form control design proposed in [45, 46] offers exponential convergence of the states to the origin and the one proposed in [47] offers stabilization to a small neighborhood of the origin.

Discrete time supervisory controllers are used in [25, 49, 50] to switch between low level continuous time-invariant feedback controllers. Each low level controller is used to track a certain straight trajectory while the high level discrete time controller ensures that the overall desired trajectory is successfully followed. An application of this technique can be found in [51, 52] for the position control of an under-actuated spacecraft using torque as the input.

The theory of a hybrid control law is proposed in [43] where multiple controllers are used to achieve asymptotic stability. Logic-based switching is performed to pick a controller that generates an appropriate control signal to achieve stabilization. This technique is helpful in the control of nonholonomic systems with parametric uncertainty.

The major disadvantage of hybrid control technique is that there is no direct way of optimizing the controller and it is hard to incorporate input and output constraints. Also, the controller switching is a discontinuous process where the switching decision is made based on the initial conditions. Since the initial conditions vary continuously, there is a hidden possibility of indecision at the boundaries of the controller selection regions.

2.1.4 Robust and Adaptive Control Laws

In [17], robust adaptive motion/force tracking control of uncertain nonholonomic Lagrangian mechanical systems with classical nonholonomic constraints is established. The Lagrangian system is classified into kinematic control and dynamic control. Lagrangian system's kinematic and dynamic equations are converted into chained form. Further transformation narrows down the tracking problem to finding the motor torque, such that trajectory tracking is successfully accomplished. The robust control law with uncertainties, such as unknown inertia values, is established. The controller consists of two parts. In the first part, an

embedded input ensures the tracking of desired trajectory if mechanical dynamics are not present. The second part of the controller regulates the contact force by utilizing torque as an input. The advantage of the proposed algorithm is that the controller has the ability to deal with uncertain systems where the model parameters are changing with time or are completely unknown and unmeasurable. The algorithm also considers input constraints. However, it ignores output constraints. Since the system is time varying, the control algorithm for tracking is computationally extensive. Design optimization is not discussed in this work.

2.1.5 Optimal Control Using Neural Networks

Neural networks are also used in controls literature to predict the future trajectory of the WMRs. In [53] , an optimal control trajectory is computed using neural networks in the presence of obstacles. However, a nonlinear model of a WMR is used for trajectory tracking and hence, adds to the computational workload. In [54], a path following problem is solved for a WMR by using neural networks for optimal control trajectory computation. The design is based on optimizing the control law to minimize the error between the actual and reference trajectory.

2.1.6 Kalman Filtering

In practical scenarios, the state values are determined by using sensors and are not accurate due to disturbances. Kalman filtering is a techniques used to account for these disturbances. It is based on a set of mathematical equations that allow estimation of the states of a process recursively while minimizing the mean of the squared error [55]. The incorporation of the state constraints is not easy with this technique, however, it has been done in [56] for

nonlinear vehicle tracking.

Kalman filtering technique can be used with different control designs to accurately predict the states of a system at any time instance. For example, it has been used with the model predictive control technique in [57] for accurately estimating the states of a WMR and in [58] for estimating the orientation and position of a WMR.

2.2 Modern Nonholonomic Control Techniques

There are three main problems with the control design solutions described above:

1. They do not allow an easy application of input and state constraints that may arise in practical situations.

2. There is no easy way to tune the final control law to obtain good transient performance.

3. Control signal optimization is not quite as systematic.

All these problems can be directly addressed by applying MPC to the WMR problem. The major difference between MPC and other classical optimal control techniques such as H_2 and H_∞ is that it solves an optimal control problem over a finite horizon as opposed to an infinite horizon. Hence, MPC is an optimal control strategy in which the control law is designed based on minimizing a cost function over a finite horizon window. Although MPC is not a new optimal control strategy, researchers have only recently started employing it to control nonholonomic WMRs. The methods of optimization applied in MPC have the distinct advantage that they allow the system to directly implement input and state constraints that may arise in practical situations. The following subsections inform the

readers of the control literature available on linear and nonlinear MPC for the control of WMRs.

2.2.1 Nonlinear Model Predictive Control

Although nonlinear MPC techniques are well developed [59, 60], researchers have started employing it only recently to control WMR mainly because of the contemporary development in processor technology. Nonlinear MPC algorithms require excessive computations and hence, faster processors are required for their implementation.

In [61], a discrete-time nonlinear MPC technique is developed for trajectory tracking of WMR. The cost function used is given by the following equation:

$$V(x,k,u) = \sum_{i=k}^{k+N-1} l(x(i),\mathbf{u}(i)) + F(x(k+N))$$

where $\mathbf{u} = u(k), u(k+1), ..., u(k+N-1)$ and $x(i) := x^u(i;(x,k))$. It depends on the future control trajectory u and the state vector x. Therefore, the minimization of this cost function reduces the error between the actual and reference trajectory using minimum control effort.

In [62], MPC is employed to minimize the following cost function:

$$J_Q(t) = \int_0^\infty [\hat{e}_t^2(\tau) + \lambda \hat{u}_t^2(\tau)]d\tau$$

where $\hat{u}_t$ and $\hat{e}_t$ are the predicted input and error trajectories over the interval $[t,\infty)$. Hence, the cost minimization is done over an infinite time horizon. In both these papers, a nonlinear WMR model is used for stabilization.

In [15], a nonlinear model of a WMR is controlled by using a nonlinear MPC algorithm in state-space form. The problems of point stabilization and trajectory tracking are solved in the paper. A novel cost function is minimized over the prediction horizon.

In [27], a nonlinear WMR is controlled to perform trajectory tracking using nonlinear MPC which leads to a non-convex optimization problem. Constrained Optimization is done in which two different cost functions are minimized using MATLAB QP algorithm and the results are compared. The standard quadratic cost function that penalizes the changes in control inputs and the tracking error show slow convergence to the desired trajectory. Hence, a modified cost function is minimized in a bid to reduce the convergence rate. The modified cost function first proposed in [15] uses exponentially increasing state weighting and the results show significant improvement in the system response in terms of convergence rate. It is shown that the nonlinear MPC is not implementable in real time for prediction horizons longer than $10\ seconds$.

2.2.2 Linear Model Predictive Control

In the absence of constraints, the MPC algorithm applied to a linear system is quite fast and is well handled by linear, quadratic and Gaussian controls [63]. The biggest hurdle in applying MPC control is that computation time may be excessive when constraints are present and does not always allow for real time control. Over the years, many control algorithms have been developed to solve a constrained MPC problem which is setup in a Quadratic Programming (QP) form. These algorithms can be classified into two major groups:

1. Active Set Methods (ASMs)

2. Interior Point Methods (IPMs)

Theoretically, ASMs require exponential and IPMs require polynomial number of iterations in the worst case to solve a QP problem [64]. In [65], however, a detailed comparison

study is conducted by applying ASMs and IPMs to randomly generated QP problems and it is concluded that ASMs are a better option in terms of computation time and number of iterations if there are only a few number of constraints and decision variables in a problem. The decision variables are computed by multiplying the number of inputs by the prediction horizon. In our case, there are only two inputs to the WMR and only two constraints that are applied to the inputs. Therefore, ASMs are a good choice to reduce the computational complexity in our case, provided the prediction horizon is kept small.

A traditional quadratic cost function that needs to be minimized over a finite prediction horizon P using MPC is given by the following equation:

$$J(k) = \sum_{i=k}^{k+P-1} \frac{1}{2}\tilde{x}(i)'Q\tilde{x}(i) + \frac{1}{2}\bar{u}[i]'R\bar{u}(i) \tag{2.2.1}$$

subject to linear constraints:

$$A(k)U \leq b(k) \tag{2.2.2}$$

where $U(k) = \begin{bmatrix} u(k)' & u(k+1)' & \dots & u(K+P-1)' \end{bmatrix}$ is the computed future control trajectory and $\bar{u}(i) = u(i) - u(i-1)$ is the difference between the current and the past controls signal values, i.e. the change in control input at each sampling instance. Q and R are diagonal matrices used for penalizing any changes in input or error variables. These matrices can be used for tuning purposes to improve the transient performance of the system.

In [11, 66], Generalized Predictive Control is used to solve the path following problem for a WMR. The control design is simplified by assuming that the input is simply the angular velocity and that the linear velocity remains constant. Hence, a single input linear model is used for the optimization problem. The optimization involves the minimization of a cost function that depends on the error between the actual and the reference trajectory.

In [67], a nonlinear model of a boat is linearized about different equilibrium points. A discrete event controller is used to switch between these linear models based on the yaw

rate. The system is then setup as a QP problem and an MPC algorithm is used to perform the dynamic path following optimization. A quadratic cost function is used which not only minimizes the error between the reference path and the actual path, but also minimizes the control effort, i.e. the rudder rate. The effects of changing MPC parameters such as the cost function matrices, the prediction horizon and the sampling time are also analyzed in detail in the report. It is shown that the simulation time required is small enough for real time implementation due to the use of the linear models.

In [27], a continuous-time nonlinear kinematic model for a nonholonomic WMR is discretized using Euler approximation. The key idea consists of using a successive linearization approach, yielding a linear time-varying description of the system [68]. Due to this linearization, the control problem becomes convex and hence, is quicker to solve than a non-convex problem faced when a nonlinear model of WMR is solved using nonlinear MPC. The linearization is performed using the Jacobian method and the linear model is converted to the state space form. The optimal trajectory tracking problem is setup as a QP problem and is solved using MATLAB's QP algorithm. The performance of the linear MPC is compared with a nonlinear MPC using the same setup. It is concluded that, in comparison with nonlinear MPC, the linear MPC has promising applications for problems which require longer prediction horizons.

Chapter 3

The Linear MPC Algorithms

In this chapter, we discuss the different algorithms used in this research along with the LMPC technique to solve a quadratic programming problem.

MPC is an optimal control strategy in which the control law is designed based on minimizing a cost function over a finite horizon window. The methods of optimization applied in MPC have the distinct advantage that they allow the system to directly implement control and state constraints that may arise in practical situations. There are three distinct advantages of using MPC as opposed to other techniques to control WMR:

1. MPC allows an easy application of input and state constraints that may arise in practical situations.

2. It is easy to tune the final control law to obtain good transient performance.

3. Control signal optimization is systematic.

MPC is not a new optimal control strategy. The major difference between MPC and other classical optimal control techniques such as H_2 and H_∞ is that it solves an optimal control problem over a finite horizon as opposed to an infinite horizon.

The biggest hurdle in applying MPC control is that computation time may be excessive when constraints are present and does not always allow for real time control. Active Set Methods are generally used to solve a constrained MPC problem which is setup in a quadratic programming form. ASMs usually require fewer variables to describe a problem, which decreases the computational workload and in turn, decreases the time required for each iteration. One of the well known ASMs algorithm is Fletcher's ASM. In Fletcher's ASM, single change to active set are made per iteration to ensure that even the intermediate solutions do not violate the input and the output constraints. This increases the number of iterations required to solve the MPC problem where a greater number of input and output constraints are present. To mitigate this problem, Non-Feasible Active Set Method (NF-ASM) is introduced [1], which further increases the speed of the Fletcher's ASM by making multiple changes to active set per iteration. Also, in Fletcher's ASM, changes to active set method are based on the magnitude of the largest constraint violation.On the other hand, NF-ASM makes left to right changes (w.r.t. time) to the set of active constraints. In other words, changes that occurred earlier in time are considered before the changes that occurred later in time, which results in implemented control being closer to the final optimal control. Both the Fletcher's Algorithm and the non-feasible algorithm explained in [1] are applied to the linearized model (1.2.2) for trajectory tracking and cost function minimization over the prediction horizon P. A quadratic cost function given below is used for optimization:

$$J(k) = \sum_{i=k}^{k+P-1} \frac{1}{2}\tilde{x}(i)'Q\tilde{x}(i) + \frac{1}{2}\bar{u}[i]'R\bar{u}(i) \tag{3.0.1}$$

subject to linear constraints:

$$A(k)U \leq b(k) \tag{3.0.2}$$

where $U(k) = \begin{bmatrix} u(k)' & u(k+1)' & \dots & u(K+P-1)' \end{bmatrix}$ is the computed future control trajectory and $\bar{u}(i) = u(i) - u(i-1)$ is the difference between the current and the future controls signals, i.e. the change in control input at each sampling instance. Q and R are diagonal matrices used for penalizing any changes in input or error variables. These matrices can be used for tuning purposes to improve the transient performance of the system.

The pseudo-codes of NF-ASM and FL-ASM are given in the following sections. For more details on these algorithms, the readers are recommended to refer to [1] and [28].

3.1 Non-feasible Active Set Method

The steps taken by the NF-ASM are summarized below. Two important variables used in the algorithm are the Lagrange multiplier L and the flagging curve F. Each constraint is given its own lagrange multiplier. An $L > 0$ indicates that the constraint is active and an L=0 indicates that the constraint is inactive. A flagging curve is used to indicate the constraints that need to be added or removed from the active set. For more details on the algorithm, it is suggested that the reader refers to [64].

NF-ASM [64]:

1. Given an initial constraint set I^0, calculate all initial conditions, including the control curve U^0, the Lagrange multipliers L^0 and the flagging curve F^0. Set the iteration number to $k = 0$.

2. Check if the control and Lagrange multipliers lead to termination conditions. If they do then terminate the algorithm with the constrained solution $U_C^* = U^k$, otherwise increment the iteration number to $k = k + 1$.

3. Choose the index i_u flagged for active set changes.

4. Separate a flagging curve $F_{i_u}^k$ that applies only to the control curve which will have active set changes.

5. Form an amended flagging curve $F_{i_u}^k$ which includes only the first segment of constraint changes that will be used on this iteration.

6. Adjust the current active set I^k based on the amended flagging curve $F_{i_u}^k$

7. Calculate the new control curve U^k and associated Lagrange multipliers L^k.

8. Set the new flagging variables F_L^k, F_B^k and F^k.

9. Return to step 2.

3.2 Fletcher's Active Set Method

The steps taken in FL-ASM as described in [64] are given below:

FL-ASM [64]:

1. Given feasible initial conditions, U^0, and I^0, set $k = 1$

2. If $\delta = 0$ does not solve

$$\min_{\delta} \frac{1}{2}\acute{\delta} H \delta + \acute{\delta} f^{(k)} \tag{3.2.1}$$

subject to $A_i^{'}\delta = 0, \quad i \in I^{k-1}$

where $f^{(k)} = f + HU^{k-1}$, then set $\tilde{I}^{k-1} = I^{k-1}$ and skip to step 4.

3. Compute Lagrange multipliers L^{k-1} and solve

$$q = arg \min_{i \epsilon I^{k-1}} L_i^{k-1} \tag{3.2.2}$$

if $L_q^{k-1} \geq 0$, terminate the algorithm with $U^* = U^{k-1}$; otherwise remove q from I^{k-1} setting the new temporary active set index to $\tilde{I}^{k-1}$;

4. Again solve

$$\min_{\delta} \frac{1}{2}\acute{\delta} H \delta + \acute{\delta} f^{(k)} \tag{3.2.3}$$

subject to $A_i^{'}\delta = 0, \quad i \in I^{k-1}$ for the direction vector $s^k = \delta$.

5. Find α^k from the maximum allowable growth,

$$\alpha^k = min(1, \min_{i:i \notin \tilde{I}^{k-1}, \acute{a}_I s^k > 0} \frac{a_i^{'} U^{k-1} - b_i}{a_i^{'} s^k}) \tag{3.2.4}$$

and set $U^{k+1} = U^k + \alpha^k s^k$

6. If $\alpha^k < 1$, add $p = \arg \alpha^k$ to $\tilde{I}^{k-1}$, and then set the new active index I^k.

7. Set $k = k + 1$ and go to step 2.

3.3 Differences Between NF-ASM and FL-ASM

The main reason why NF-ASM works faster than FL-ASM is that NF-ASM makes multiple changes to the active set per iteration. For example, in the best case scenario, if there are 25 constraint violations for a computed control curve U, it would take FL-ASM at least 25 iterations to solve the problem (one active set change per iteration). On the other hand, NF-ASM can potentially solve this problem in only one iteration. Table 3.1 lists the differences between the FL-ASM and NF-ASM [1].

As an example, let us consider a system with two inputs. The interleaved unconstrained control curve computed for this problem is given in Fig. 3.1. Figs. 3.2 and 3.3 show the individual control curves u_1 and u_2. It is clear from Fig. 3.2 that control curve u_1 violates

Table 3.1: Differences between FL-ASM and NF-ASM

FL-ASM	NF-ASM
One active set change per iteration	Multiple active set changes per iteration (for example, both upper and lower bound constraint violations are considered at every iteration)
Changes made based on the magnitude of the largest constraint violation	Changes made from left to right w.r.t time (i.e. the constraints that are violated earlier in time are considered first)
One active set change per iteration	Changes performed in blocks (i.e. if multiple connected samples of the control curve violate the constraints, the active set is modified to incorporate all of those segments)
All intermediate solutions obtained at the end of each iteration must satisfy all constraints	Intermediate solutions obtained at the end of each iteration may not satisfy all constraints (i.e. infeasible solutions w.r.t. constraints may be obtained)

the upper bound constraint during the time interval $1 - 3\ seconds$ and the lower bound constraint during the interval $3 - 4\ seconds$. On the other hand, Fig. 3.3 shows that control curve u_2 violates the upper bound constraint during the time interval $2 - 3\ seconds$ and it does not violate the lower bound constraint. With FL-ASM, we would make changes to the active set based on the magnitude of the highest constraint violation. The highest constraint violation occurs in u_2 during the time interval $2 - 3\ seconds$. Hence, only this part of the control signal will be modified in the next iteration. On the other hand, the NF-ASM makes changes from left to right with respect to time and starts with the constraint violations that occur earlier in time. The first upper bound and lower bound constraint violations occur in signal u_1. Hence, the active set will be changed based on these constraint violation. The upper bound constraint violation for the control signal u_2 will not be considered for active set changes in the current iteration. Hence, a non-feasible solution may be obtained at the

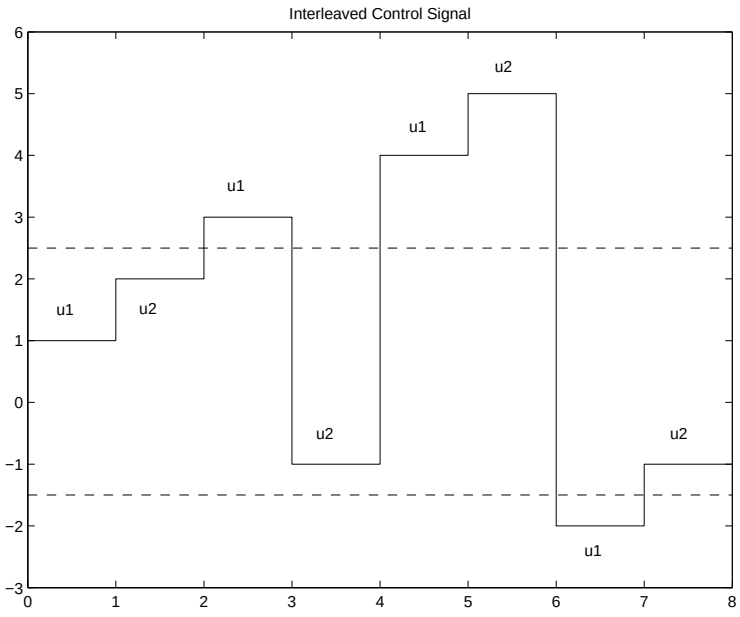

Figure 3.1: Example of an Interleaved Control Curve

end of iteration since the constraint violation of u_2 is not dealt with. Also, since the upper bound violations of u_1 during $1-2\ seconds$ and $2-3\ seconds$ occur consecutively (i.e. the two violations are "connected" to each other), they will be considered as one block and changes to them will be made in a single iteration. This process of making changes in blocks as opposed to a single active set change per iteration is the primary reason why NF-ASM works faster that FL-ASM.

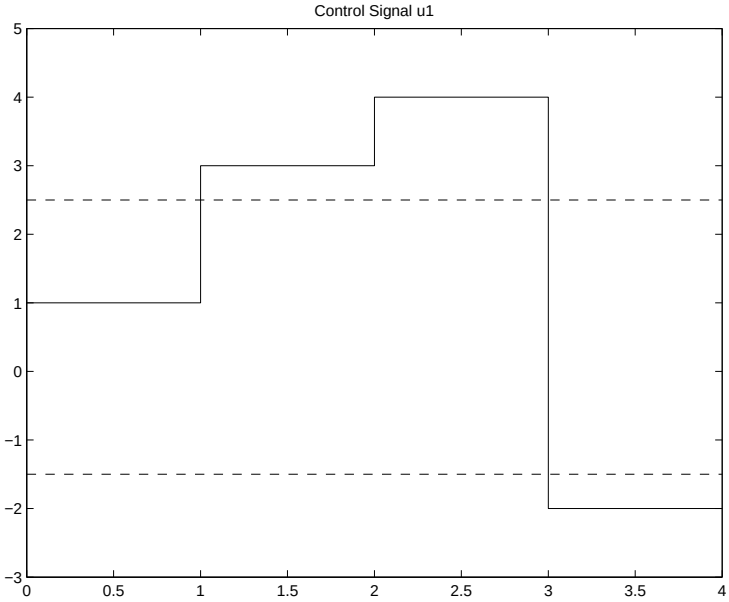

Figure 3.2: Control Input u_1

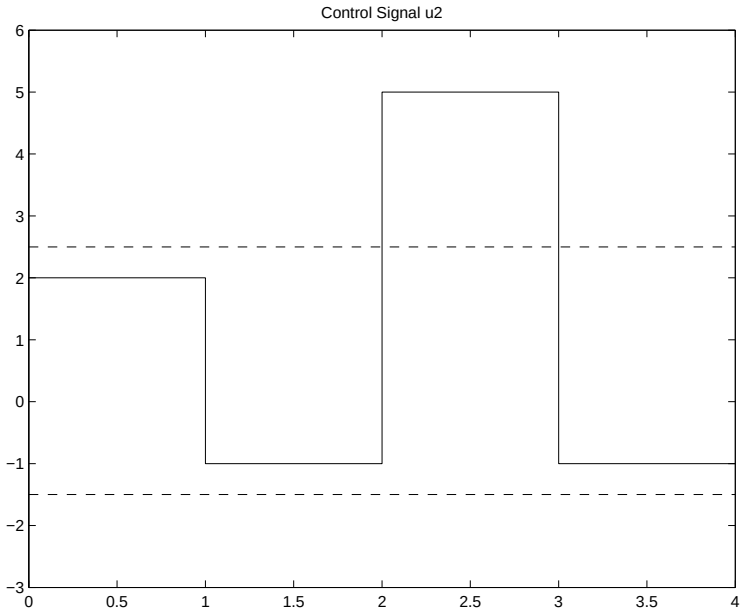

Figure 3.3: Control Input u_2

Chapter 4

MPC Simulations for the WMR

In this Chapter, the overview of the applied technique to tracking a trajectory using WMR is presented. Also, the performance metrics used to compare the performance of the different algorithms used in linear MPC technique are detailed. Different reference trajectories are also generated using MATLAB's ordinary differential equations solver ($ODE45$) on the model $(1.2.1)$ and the results are discussed. Finally, different simulation setup used to gather results are listed and the rationale behind using these setups is explained.

4.1 The Overall Control Structure

For a trajectory tracking problem, a reference trajectory is generated by solving the system of differential equations given in $(1.2.1)$. The difference of the reference trajectory and the actual trajectory of the WMR (i.e. the trajectory error) is fed to the MPC. The MPC generates an optimized control input trajectory. Since receding horizon MPC is used, therefore, only the first part of the optimized control trajectory is fed to the linearized WMR model to calculate the output trajectory of the WMR. It is important to note that the linear WMR model described in $(1.2.2)$ is time-variant. Hence, a feedback loop structure is used where the the control input to the MPC changes based on the changing system and input matrices

$A(t)$ and $B(t)$. The overall control structure is given in Fig. 4.1.

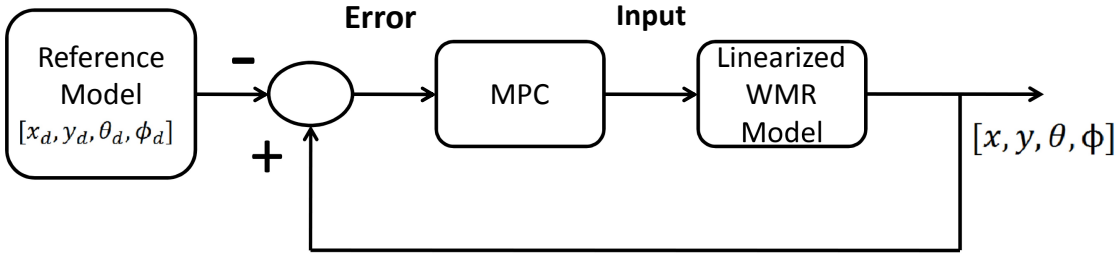

Figure 4.1: The Overall Control Scheme

4.2 System Specifications for Simulations

For all simulations, a standard Laptop PC (Toshiba M700) is used with Intel(R) Core 2 Duo CPU, 2.20 GHz, 2.00 GB RAM and a 32-bit Windows 7 Operating System. It is important to note that the time measurements in the simulations are done for comparison purposes only. Generally, time measurements are variable and depend strongly on the CPU status. Hence, each simulation is conducted 25 times and the average value of time is presented as the final result in the next Chapter.

4.3 Performance Metrics

In order to compare the performance of NF-ASM with FL-ASM and MATLAB's QP algorithm, three performance metrics are used:

1. **Total cost value** gives a measure of the quality of solution. The tracking problem is setup as a minimization problem. Hence, the lower the cost, the better the quality of solution. The cost function penalizes both the tracking error and the input changes. Hence, by minimizing the cost, we are actually minimizing the control effort and the

state errors to achieve better tracking performance. MPC allows us to give precedence to either tracking error reduction or control input minimization by changing the relative weighting in the cost function.

2. **Time per iteration** gives a measure of the speed of the algorithm. This parameter is important because it tells us whether an algorithm can solve a given problem in real-time.

3. **Number of iterations** (or **Total time consumed**) to converge to a solution gives a measure of the computational workload and the overall convergence speed of the algorithm.

To ensure unbiased results, exactly the same values for MPC tuning parameters for each simulation set are used. These parameters include the prediction horizon, $Q = q \times I_s$ and $R = r \times I_i$ (where I_s is the identity matrix of size 4×4 and I_i is the identity matrix of size 2×2).

4.4 The Trajectory Tracking Process

A continuous-time trajectory $[x_d \ y_d \ \theta_d \ \phi_f]$ is generated using the process explained in the last section. This trajectory is discretized and a linear MPC algorithm is then used iteratively to track the reference trajectory on a point-by-point basis.

An example reference trajectory is shown is Fig. 4.2. The points labeled $A - E$ are obtained through the discretization process using a certain sampling rate and are successively tracked from left to right using linear MPC to achieve the overall trajectory tracking objective. It is interesting to note that if the sampling rate is increased, we essentially have less gap between the points $A - E$. Hence, by using a higher sampling rate, we ensure

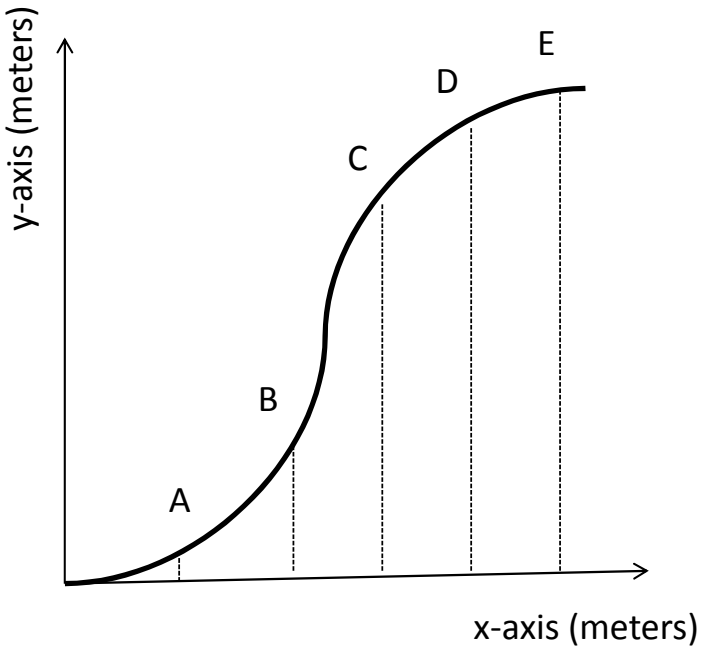

Figure 4.2: A Sample Reference Trajectory

that all the dynamics of the systems are captured when the continuous to discrete time conversion is conducted. However, this essentially adds more points to be tracked in our case; in essence, slowing down the overall trajectory tracking process. On the other hand, the trajectory tracking process can be sped up by reducing the sampling rate; which, of course, leads to an increase in the tracking error. Therefore, a balance between the tracking error and the speed of the tracking needs to be achieved which entirely depends on the application. The reference trajectory is generated and discretized off-line in our case and hence, its sampling rate can be changed based on the needs. The overall trajectory tracking process is given in the flowchart in Fig. 4.3.

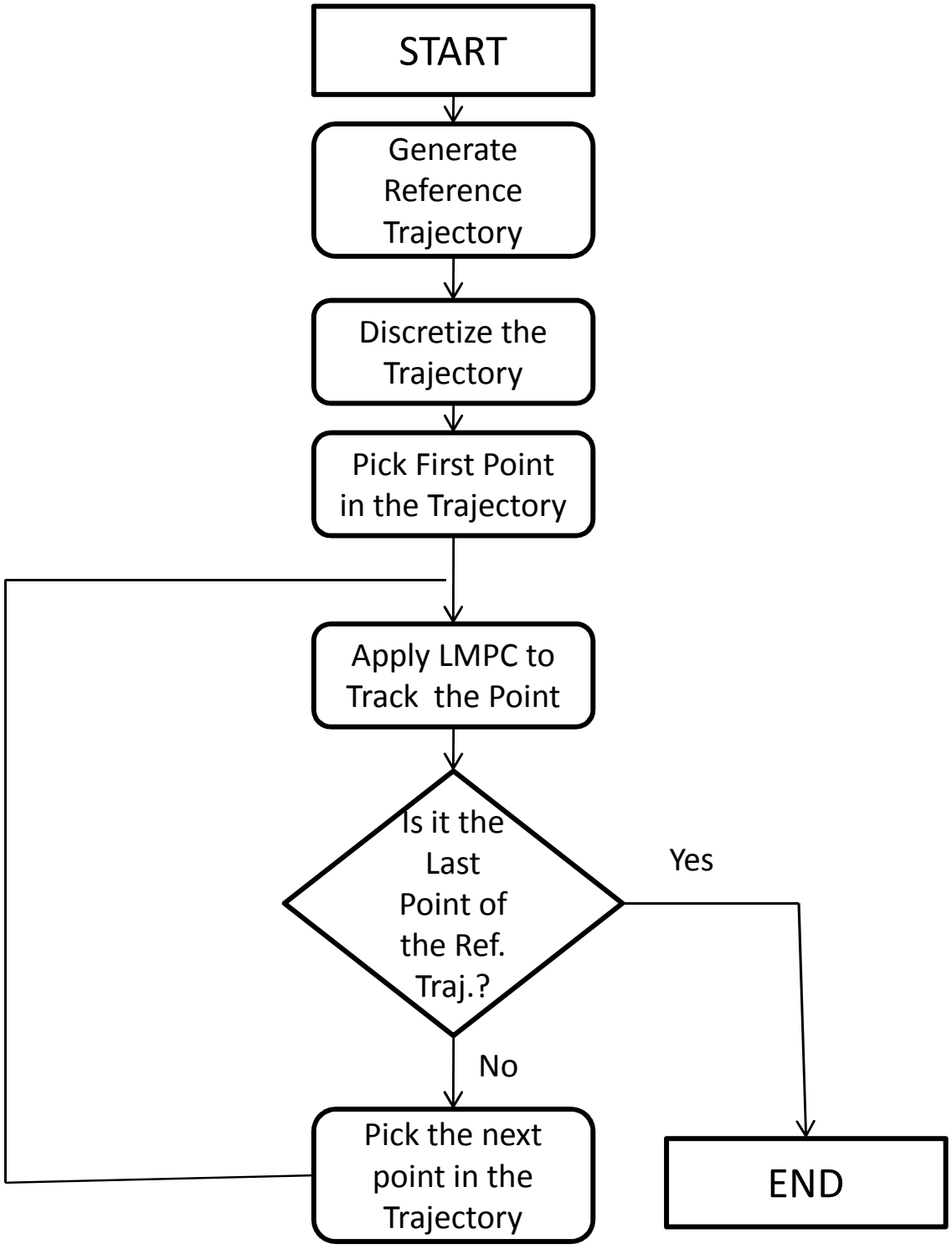

Figure 4.3: The Overall Trajectory Tracking Process

4.5 Reference Trajectory Generation

The first step in trajectory tracking is to generate the reference trajectory using the non-linear system of differential equations given in $(1.2.1)$.

The reference trajectory is generated using $ODE45$ function in MATLAB. The reference WMR model in (1.2.1) is used for the simulation. The model is re-iterated below for convenience [26] :

$$\begin{aligned} \dot{x_d}(t) &= v_{d1}(t)\cos\theta_d(t) \\ \dot{y_d}(t) &= v_{d1}(t)\sin\theta_d(t) \\ \dot{\theta_d} &= v_{d1}(t)\frac{\tan\phi_d(t)}{l} \\ \dot{\phi_d} &= v_{d2}(t) \end{aligned} \tag{4.5.1}$$

where $(x_d(t), y_d(t))$ represents the desired position of the robot, $\theta_d(t)$ denotes the desired orientation of the WMR with respect to the horizontal axis, $\phi_d(t)$ is the desired steering angle and l is the distance between the two axles of the car as shown in Fig. 1.2. The control inputs to (1.2.1) are the desired linear velocity of the WMR, v_{d1}, and its desired steering velocity, v_{d2}. The $\begin{bmatrix} x_d & y_d & \theta_d & \phi_d \end{bmatrix}$ in the above representation completely define the desired location and orientation of a WMR at a particular instance in time.

4.5.1 Generating the Circular Reference Trajectory

For circular trajectory generation using the above reference model, the inputs $v_{d1} = R\omega$, $v_{d2} = 0$ and the initial conditions $\begin{bmatrix} x_d & y_d & \theta_d & \phi_d \end{bmatrix}$ = $\begin{bmatrix} 0 & 0 & 0 & \arctan\frac{R}{l} \end{bmatrix}$ are used [26]. R is the radius of the inscribed circular path and ω is the angular velocity.

For the simulation, we use $R = 1\ m$, $\omega = 1\ rad/s$ and $l = 1\ m$ for the reference trajectory. The output of simulation is shown in Fig. 4.4. It is important to note that the

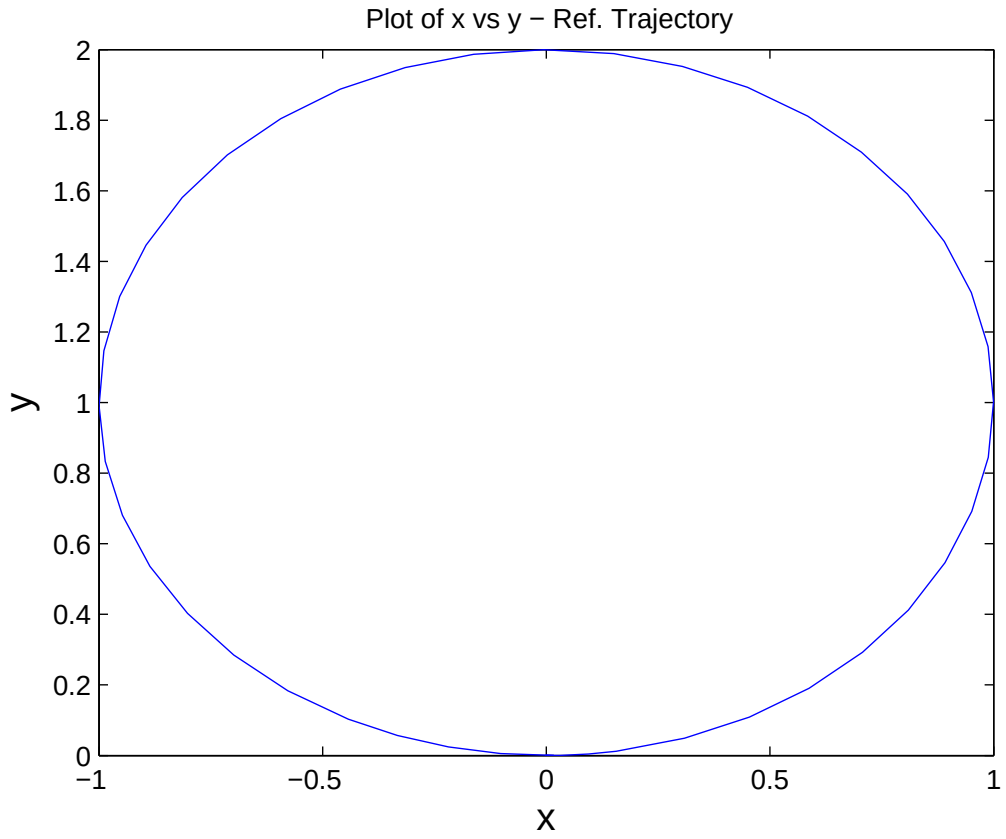

Figure 4.4: Circular Reference Trajectory

circular trajectory generated here is essentially a set of point in cartesian form that are connected together to form a circular path shown in Fig. 4.4. We use the MPC algorithm on the linearized WMR model to track these sets of points one-by-one.

4.5.2 Generating the Square Reference Trajectory

We generate the square reference trajectory using ODE45 in MATLAB using the model (1.2.1). The sides of the square are formed by using a combination of vertical and horizontal lines and the vertices of the square are replaced by parts of a circle. Fig. 4.5 shows the generated reference trajectory divided into different parts. In order to generate each part

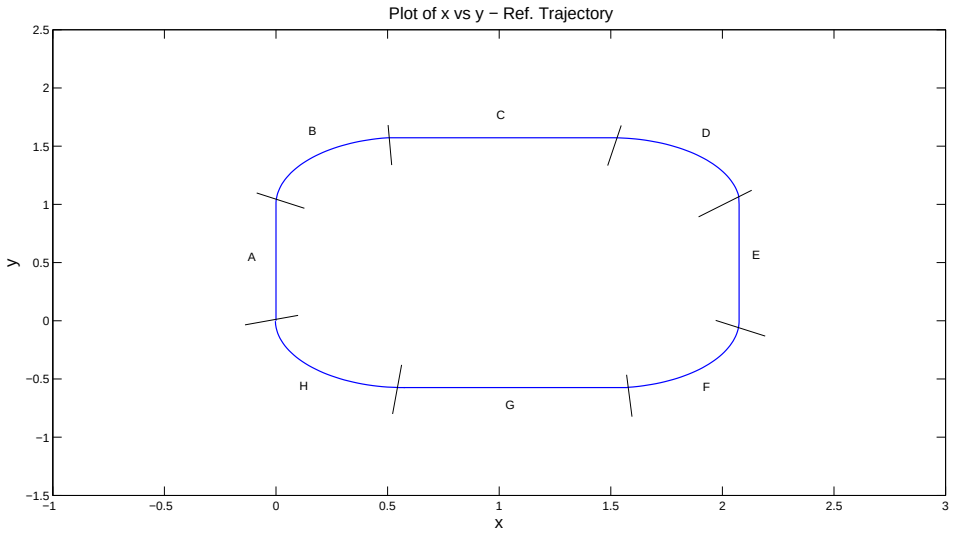

Figure 4.5: The Square Trajectory Components

of the reference trajectory, we require different initial conditions for $[x_d\ y_d\ \theta_d\ \phi_d]$. These values are given in Table 4.1. Note that for straight parts of the square trajectory, i.e. A, C, E and G, the steering angle is kept as zero where as for curved trajectories, i.e. B, D, F and H, the steering angle assumes a non-zero value. It is important to realize that the square trajectory generated here is essentially a set of points in cartesian form that are connected together to form a square path shown in Fig. 4.5. We use the MPC algorithm on the linearized WMR model to track these sets of points one-by-one.

4.6 Simulation Sets

A discrete-time linear time-varying model of a general WMR is used for the simulation. The two inputs to this linear model are the difference in the actual and desired linear velocities $(v_1 - v_{d1})$ and the difference in the actual and desired steering velocities $(v_2 - v_{d2})$. In this research, we put constraints on these control inputs. In other words, we put constraints

Table 4.1: Initial Conditions for Square Trajectory Generation

Traj. Component	Initial Condition $[x_d\ y_d\ \theta_d\ \phi_d]$	$ODE45$ Runtime (seconds)
A	$[0\ 0\ \pi/2\ 0]$	1
B	$[0\ 1\ \pi/2\ -\pi/3]$	0.83
C	$[0.5\ 1.5\ 0\ 0]$	1
D	$[1.5\ 1.5\ 0\ -\pi/3]$	0.83
E	$[2\ 1\ -\pi\ 0]$	1
F	$[2\ 0\ -\pi/2\ -\pi/3]$	0.83
G	$[1.5\ -0.5\ -\pi\ 0]$	1
H	$[0.5\ -0.5\ -\pi\ -\pi/3]$	0.83

on the deviation of the linear and steering velocities from their reference values. The control optimization is done to minimize the control effort (i.e. to minimize the magnitude of control inputs required to solve the tracking problem) and the tracking error. MPC's inherent ability to incorporate input constraints and perform control optimization makes this possible in a very simple and systematic manner.

Different constraint settings for the inputs to the linearized WMR model in (1.2.2), i.e. $(v_1 - v_{d1})$ and $(v_2 - v_{d2})$, are used to test the trajectory tracking performance of NF-ASM. The purpose of using different constraints is to see if the algorithm performance is dependent on the size of the constraints. The performance of NF-ASM is also compared with the tracking results obtained by using FL-ASM and the QP algorithm.

For circular trajectory tracking, we use three different sets of constraints $Cc1 - Cc3$. The MPC tuning parameter values are $q = 1$, $r = 0.07$, $P\ (Prediction\ horizon) = 10s$ and $Tsim\ (Simulation\ time) = 15s$. We start by using the constraint set $Cc1$ which is not as tight as the other two constraint sets $Cc2$ and $Cc3$. The purpose of using this constraint set is to see the performance of the algorithm when the constraints do not become active as often during the simulation. We then use constraint set $Cc2$ to test the algorithm. This

Table 4.2: Constraint Sets for Circular Trajectory Tracking

Constraint Set	Constraints			
	lower bounds		upper bounds	
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)
Cc1	-1	-1	1	1
Cc2	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}
Cc3	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}

constraint set represents a more realistic situation in which the constraints stay active for a sufficient amount of time for good algorithm performance testing. Finally, $Cc3$ is used which is the tightest set of constraints for circular trajectory tracking. This constraint set may not often be used in practical scenarios but it gives a good measure of effectiveness of the algorithm under extreme constraints. Table 4.2 gives the values of each of these constraint sets.

For square trajectory tracking, we use five different sets of constraints $Cs1 - Cs5$. The MPC tuning parameters used with all three algorithms are $q = 1$, $r = 0.07$, $P\ (Prediction\ horizon) = 10s$ and $Tsim\ (Simulation\ time) = 3s$. We start by using the constraint set $Cs1$ which is not as tight as the other constraint sets $Cs2 - Cs5$. The purpose of using this constraint set is to see the performance of the algorithm when the constraints do not become active as often during the simulation. We then use constraint sets $Cs2 - Cs4$ to test the algorithm. These constraint sets represent a more realistic situation in which the constraints stay active for a sufficient amount of time for good algorithm testing. Finally, $Cs5$ is used which is the tightest set of constraints for square trajectory tracking. This constraint set may not often be used in practical scenarios but it gives a good measure of effectiveness of the algorithm under extreme constraints. Table 4.3 gives the values of each of the constraint sets used in square trajectory tracking.

Table 4.3: Constraint Sets for Square Trajectory Tracking

Constraint Set	Constraints			
	lower bounds		upper bounds	
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)
Cs1	-1	-1	1	1
Cs2	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}
Cs3	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}
Cs4	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}
Cs5	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}

Chapter 5

Results and Discussions

In this chapter, we present and discuss the results obtained from different simulation setups explained at the end of the previous Chapter. NF-ASM is applied to the WMR to track various types of trajectories and its performance in terms of speed and computational workload is compared to the performance of FL-ASM and MATLAB's Quadratic Programming (QP) algorithm. The two inputs to the linear WMR model in (1.2.2) are the difference in the actual and desired linear velocities $(v_1 - v_{d1})$ and the difference in the actual and desired steering velocities $(v_2 - v_{d2})$. We put different constraints on these control inputs. In other words, we put constraints on the deviation of the linear and steering velocities from their reference values. The purpose of using different constraints is to see if the performance of the algorithm is dependent on the size of the constraints.

5.1 Circular Trajectory Tracking

In this section, simulations are conducted to track a circular trajectory using NF-ASM. The performance of NF-ASM is compared to the performance of FL-ASM and MATLAB's QP algorithm. We apply MPC to the WMR using three different sets of constraints $Cc1 - Cc3$ explained in the pervious Chapter and present our results in the following subsections.

5.1.1 Simulations under Constraint Set Cc1

We start with relaxed constraints on the inputs, i.e. Cc1: $-1\ m/s \leq v_1 - v_{d1} \leq 1\ m/s$ and $-1\ rad/s \leq v_2 - v_{d2} \leq 1\ rad/s$. NF-ASM demonstrates good results in terms of trajectory tracking performance as shown in Fig. 5.1(a). None of the constraints become active during the simulation, as shown in Fig. 5.1(d), because the constraints on each input are too relaxed. Hence, unconstrained control optimization is performed and no active set algorithm is required to solve the tracking problem, leading to zero iterations used as shown in Fig. 5.1(c). The state errors are shown in Figs. 5.1(e-h). As seen in these figures, the state errors do not stay at a zero value. They increase in a periodic manner because the model of the WMR is time-varying, leading to a change in dynamics of the WMR. This change in system dynamics with time results in a need for the re-execution of MPC. With each change in the dynamics of the WMR model, the MPC algorithm re-stabilizes the system causing the error to reach a zero value as seen in the figures. The control inputs are shown in Figs. 5.1(i-j).

We also apply FL-ASM and MATLAB's QP algorithm to the WMR using the same simulation setup and the constraint set $Cc1$. The results of these simulations are shown in Figs. 5.2 and 5.3. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.2(a) and 5.3(a).

In FL-ASM simulation, none of the constraints become active during the simulation, as shown in Fig. 5.2(d), because the constraints on each input are too relaxed as in the case of NF-ASM simulation. Hence, unconstrained control optimization is performed and no active set algorithm is required to solve the tracking problem, leading to zero iterations used as shown in Fig. 5.2(c).

Table 5.1 summarizes our results for simulations under the constraint set $Cc1$. All algorithms give the same cost function value (i.e. $J = 1.73 \times 10^{-8}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased

Table 5.1: Circular Trajectory Tracking Results under the Constraint Set $Cc1$

Algorithm	Constraints				Total Cost ($J \times 10^{-8}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-1	-1	1	1	1.73	0	1.26
FL-ASM	-1	-1	1	1	1.73	0	1.26
MATLAB's QP	-1	-1	1	1	1.73	—	6.44

comparison of their performance in terms of computational workload and simulation time.

NF-ASM and FL-ASM algorithms take same amount of simulation time because both algorithms are performing unconstrained optimization which is exactly the same for both algorithms. MATLAB's QP algorithm, on the other hand, is approximately 5 times slower to solve the trajectory tracking problem. Also, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.1(b) and 5.2(b) but it is higher for MATLAB's QP algorithm as seen in Fig. 5.3(b).

Figs. 5.1(e-h), 5.2(e-h) and 5.3(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and MATLAB's QP algorithm, respectively. There is a relatively large error magnitude at the beginning of the iterations but the error reduces significantly in the later parts of the iterations which shows that all three algorithms are capable of solving the trajectory tracking problem.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.1(i-j), 5.2(i-j) and 5.3(i-j) , respectively. Since, the control inputs do not exceed the constraint limits imposed in $Cc1$, unconstrained control optimization is performed by all three algorithms.

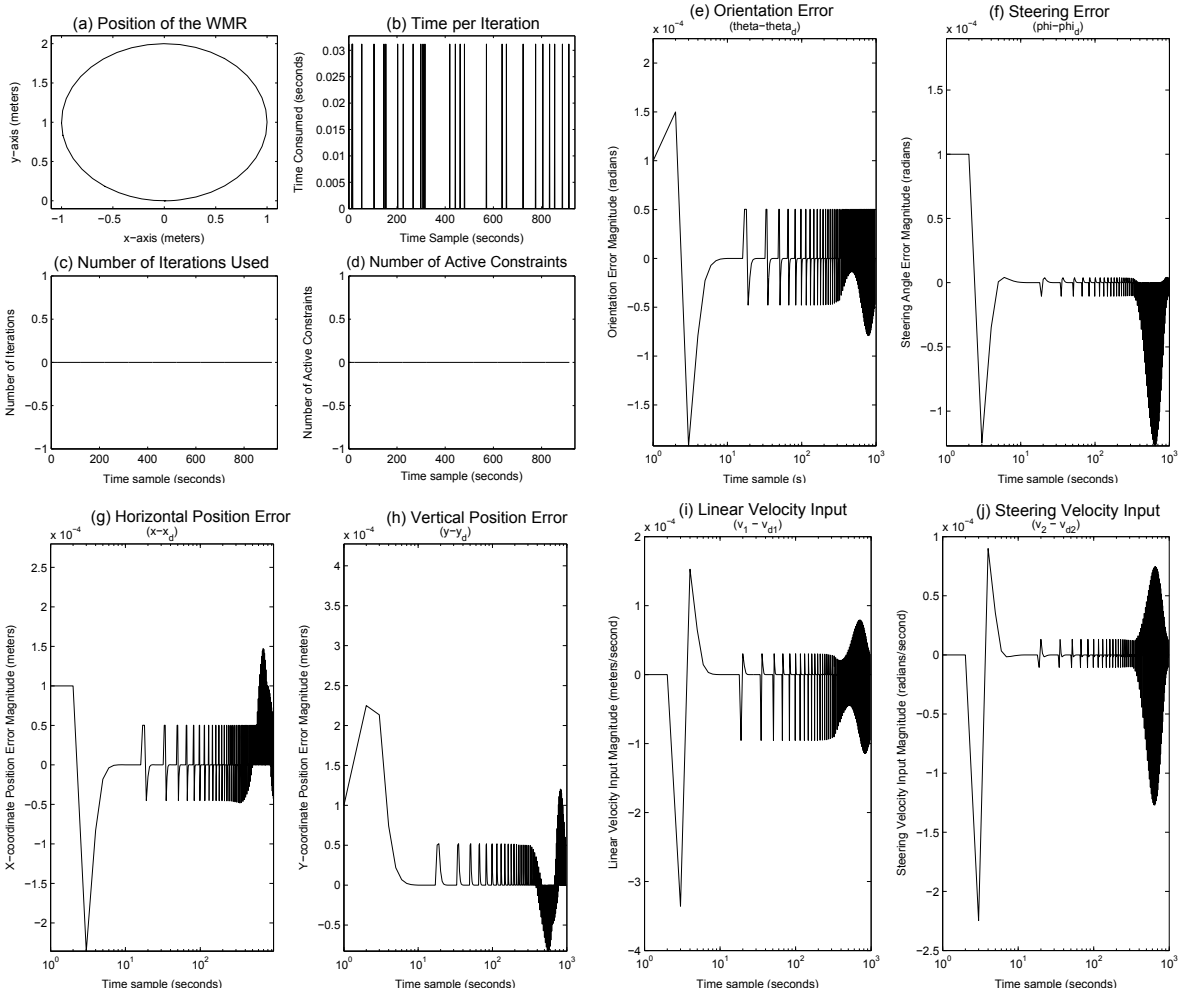

Figure 5.1: NF-ASM performance under constraint Cc1

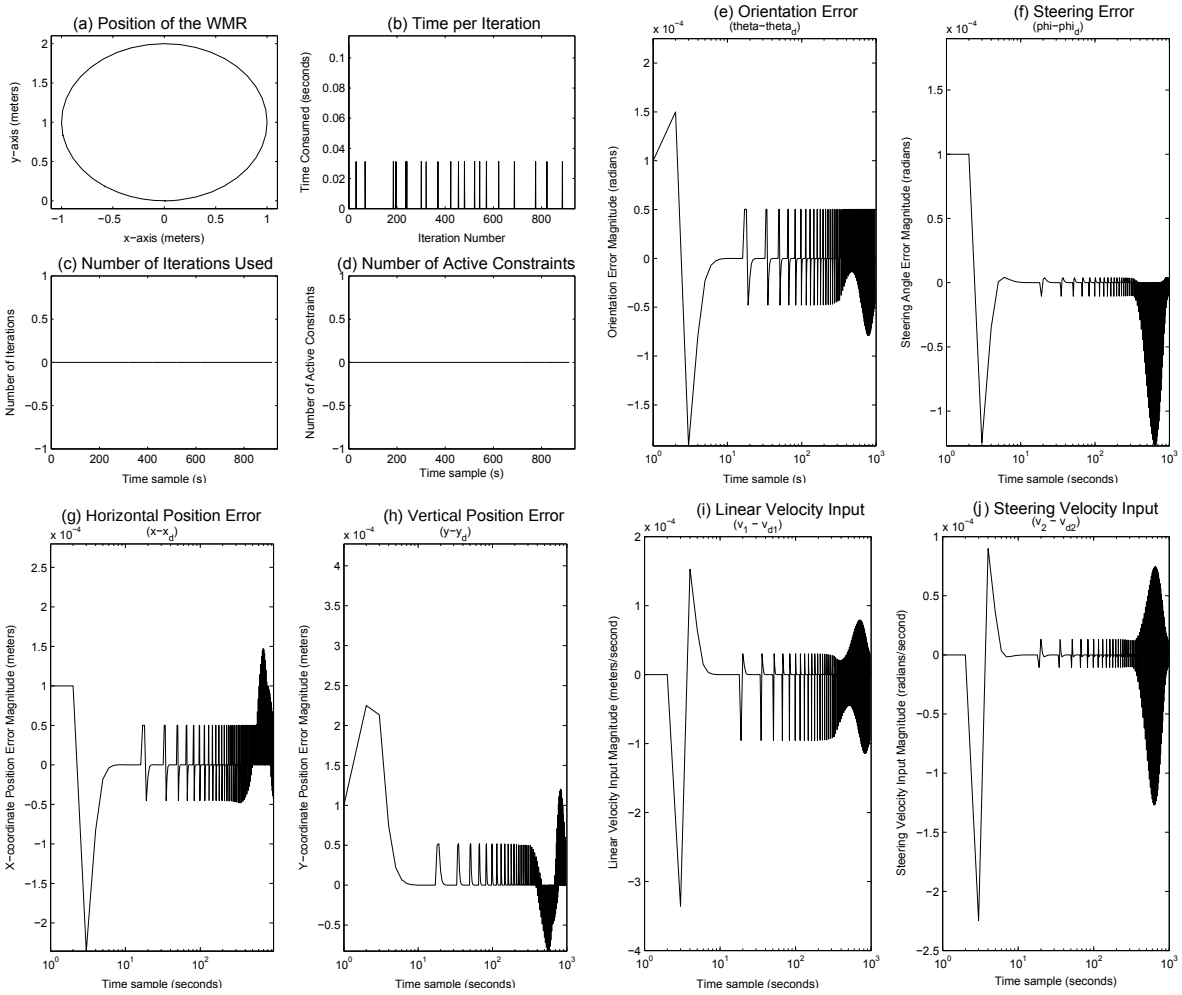

Figure 5.2: FL-ASM performance under constraint Cc1

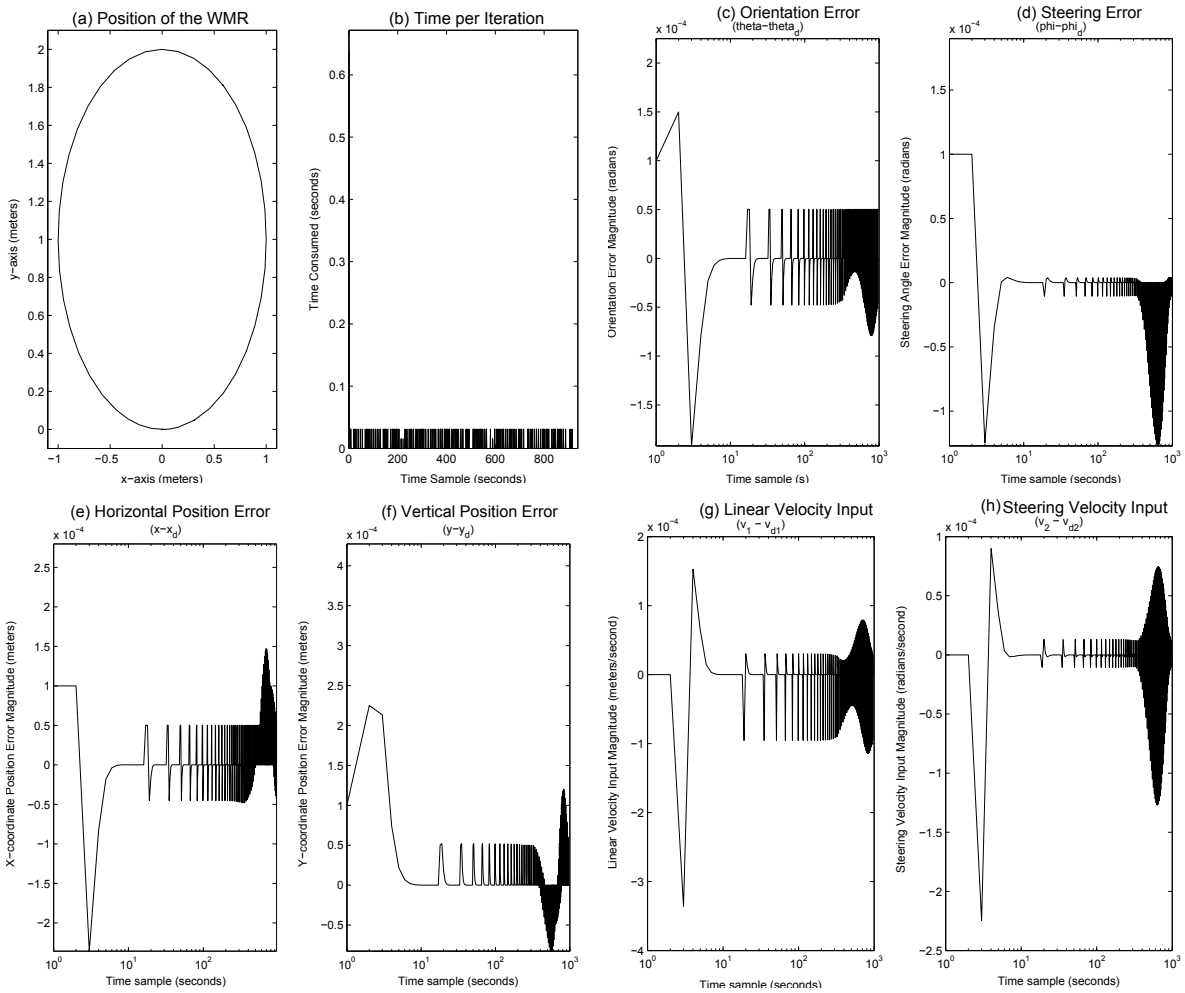

Figure 5.3: MATLAB's QP performance under constraint Cc1

5.1.2 Simulations under Constraint Set Cc2

We tighten our constraints to make them active in our second simulation, i.e. Cc2: $-1\times10^{-5}\ m/s \leq v_1 - v_{d1} \leq 1 \times 10^{-5}\ m/s$ and $-1 \times 10^{-5}\ rad/s \leq v_2 - v_{d2} \leq 1 \times 10^{-5}\ rad/s$. The tracking results are shown in Fig. 5.4(a) along with the state errors shown in Figs. 5.4(e-h). The number of active constraints at a given point toggles between zero and two as seen in Fig. 5.4(d). This shows that both input constraints are active at the beginning of every tracking step and become inactive when the WMR approaches the desired set-point. The reason is that less control effort, i.e. lower magnitude of inputs $(v_1 - v_{d1})$ and $(v_2 - v_{d2})$, is required at each time sample as we reach our desired set-point value making the input constraints inactive. The number of iterations used at each time sample are shown in Fig. 5.4(c). An average of 25 iterations per sample with 48 iterations per sample in the worst case and a total of 7545 iterations of NF-ASM are required for circular trajectory tracking under this set of constraints. The control inputs to the linearized WMR model are shown in Figs. 5.4(i-j). It is clear from the figures that the control inputs stay within the constraints $Cc2$.

We also apply FL-ASM and MATLAB's QP algorithm to the WMR using the same simulation setup and the constraint set $Cc2$. The results of these simulations are shown in Figs. 5.5 and 5.6. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.5(a) and 5.6(a).

Table 5.2 summarizes our results for simulations under the constraint set $Cc2$. All algorithms give the same cost function value (i.e. $J = 8.91 \times 10^{-8}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased comparison of their performance in terms of computational workload and simulation time. FL-ASM takes slightly less amount of time to complete the tracking as compared to NF-ASM. MATLAB's QP algorithm, on the other hand, is approximately two times slower to solve the trajectory tracking problem. Also, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.4(b) and 5.5(b) but it is higher for QP algorithm as seen in Fig. 5.6(b).

The number of iterations required to solve the trajectory tracking problem using FL-ASM is

Table 5.2: Circular Trajectory Tracking Results under the Constraint Set $Cc2$

Algorithm	Constraints				Total Cost ($J \times 10^{-8}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	8.91	7545	7.25
FL-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	8.91	5577	6.36
MATLAB's QP	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	8.91	—	13.2

slightly smaller than the number of iterations required using NF-ASM as shown in Table 5.2. This is one of the main reasons why FL-ASM is faster than NF-ASM for this scenario. Note that the constraints on the inputs are active for roughly half of the time with the constraint set $Cc2$ applied to the WMR as shown in Figs. 5.4(d) and 5.5(d).

Figs. 5.4(e-h), 5.5(e-h) and 5.6(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and QP algorithm, respectively. There is relatively large error magnitude at the beginning of the iterations but the error reduces significantly in the later parts of the iterations which shows that all three algorithms are capable of solving the trajectory tracking problem under the constraint set $Cc2$.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.4(i-j), 5.5(i-j) and 5.6(i-j) , respectively. As seen in these figures, the control inputs do not exceed the constraint limits imposed in $Cc2$.

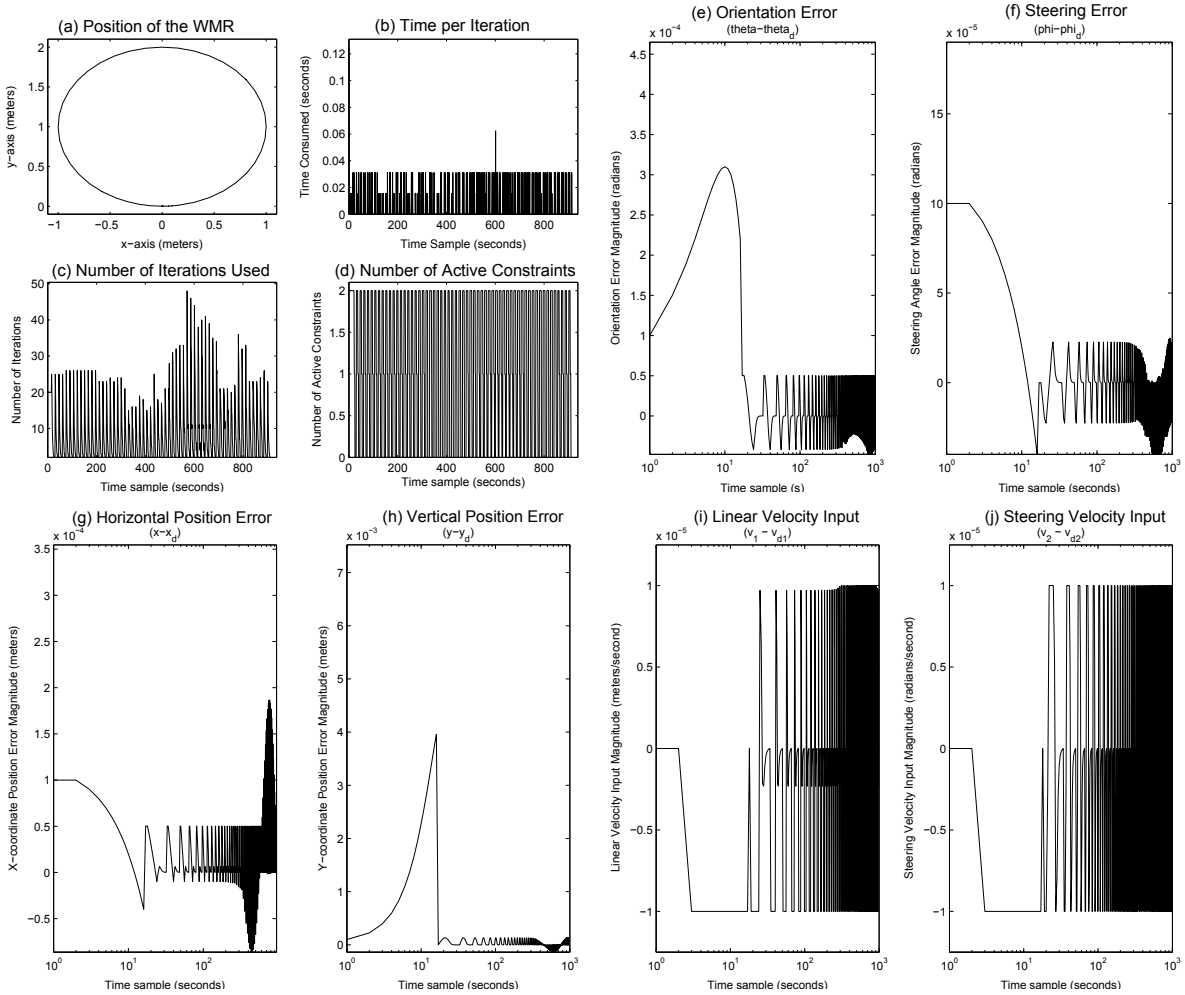

Figure 5.4: NF-ASM performance under constraint Cc2

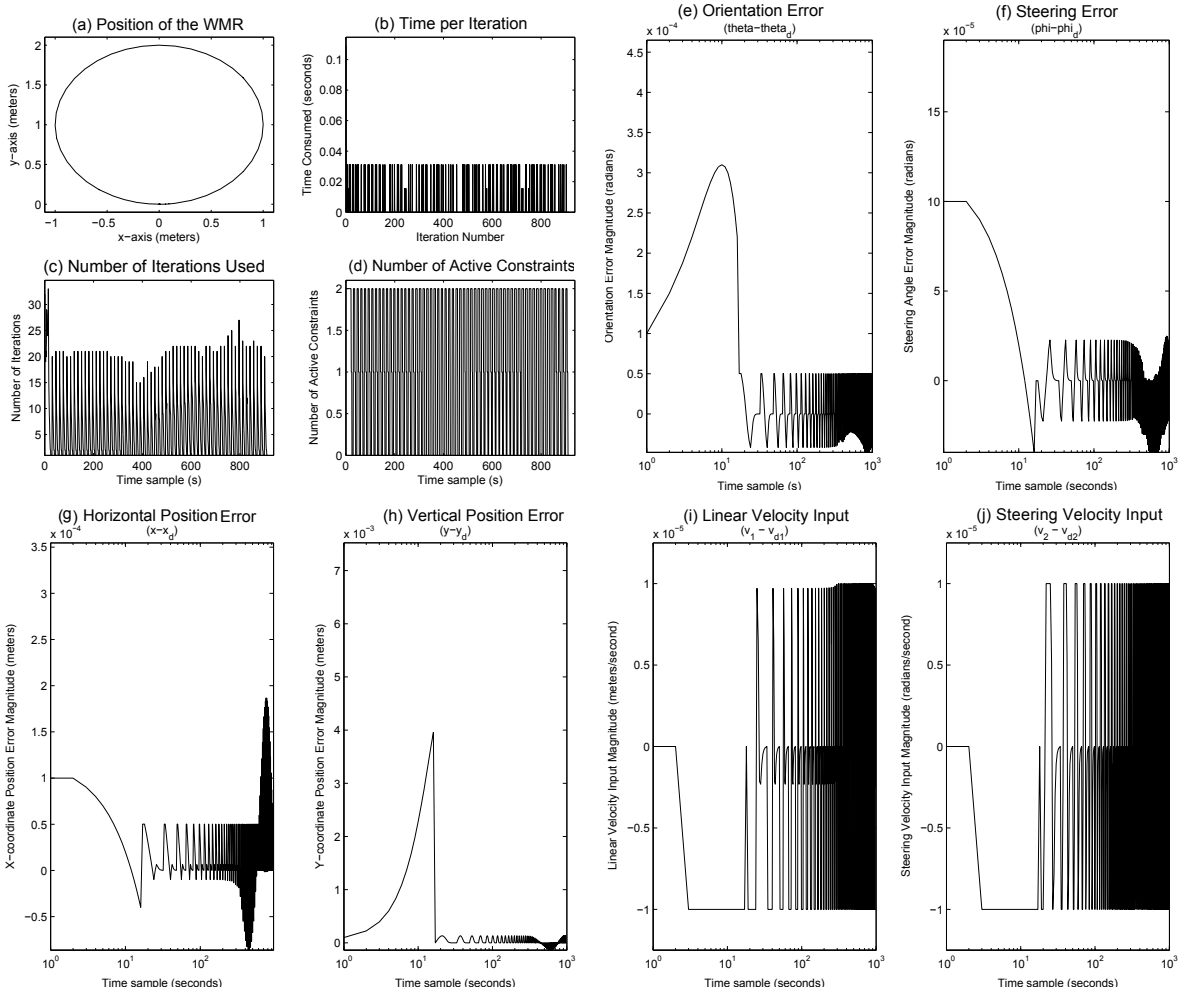

Figure 5.5: FL-ASM performance under constraint Cc2

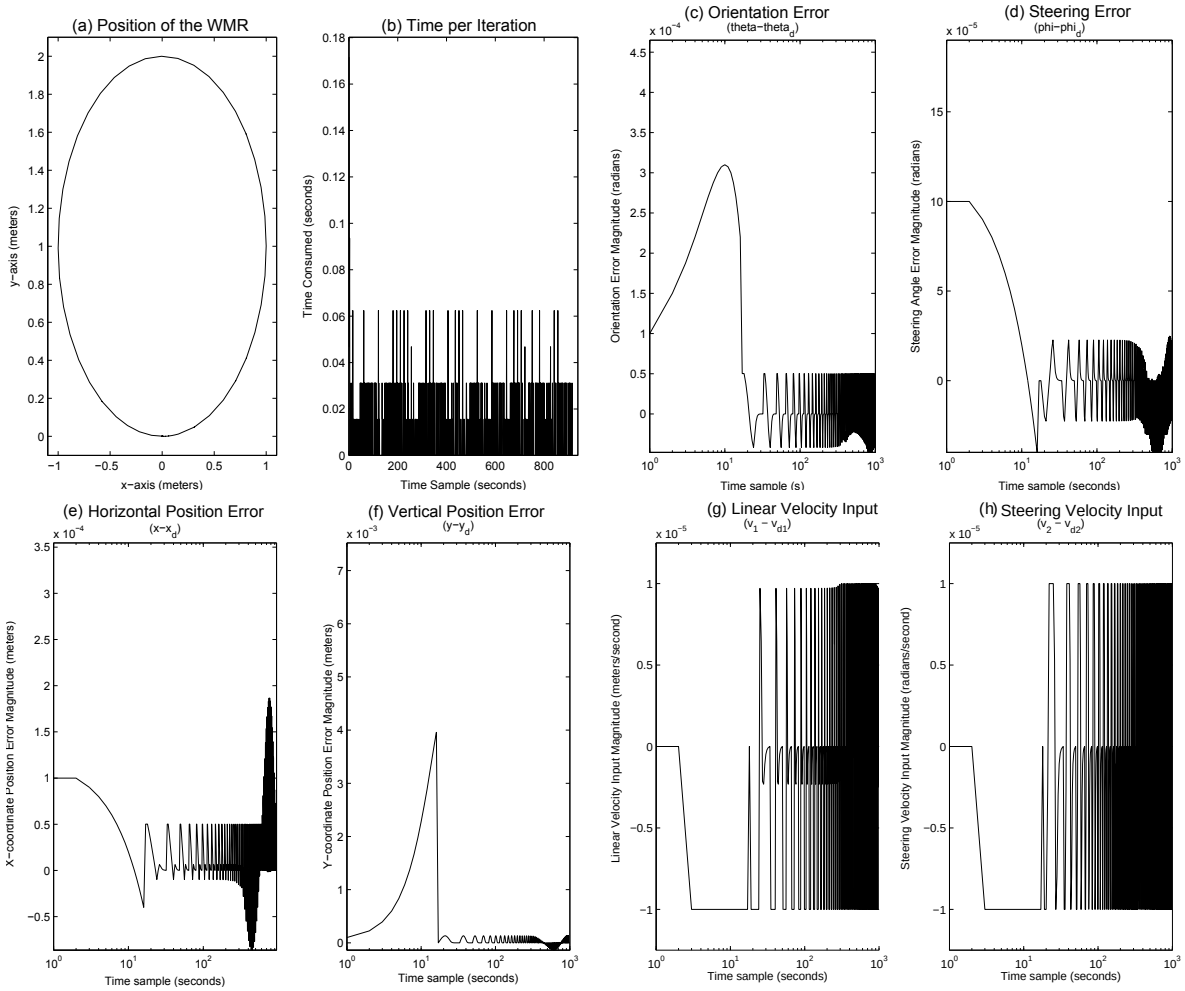

Figure 5.6: MATLAB's QP performance under constraint Cc2

5.1.3 Simulations under Constraint Set Cc3

In the third simulation, we use the constraint set Cc3: $-1 \times 10^{-6}\ m/s \leq v_1 - v_{d1} \leq 1 \times 10^{-6}\ m/s$ and $-1 \times 10^{-6}\ rad/s \leq v_2 - v_{d2} \leq 1 \times 10^{-6}\ rad/s$. The tracking results are shown in Fig. 5.7(a) along with the state errors shown in Figs. 5.7(e-h). Both constraints are active at all time during the simulation as seen in Fig. 5.7(d). The number of iterations used at each time sample are shown in Fig. 5.7(c). The control inputs to the linearized WMR model are shown in Figs. 5.7(i-j). It is clear from the figures that the control input stays within the constraints $Cc3$.

As in the previous cases, we also apply FL-ASM and MATLAB's Quadratic Programming algorithm to the WMR using the same simulation setup and the constraint set. The results of these simulations are shown in Figs. 5.8 and 5.9. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.8(a) and 5.9(a).

Table 5.3 summarizes our results for simulations under the constraint set $Cc3$. All algorithms give the same cost function value (i.e. $J = 1.51 \times 10^{-6}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased comparison of their performance in terms of computational workload and simulation time. NF-ASM consumes less number of iterations as compared to FL-ASM. Note that the constraints on the inputs are active for all times during the simulation as shown in Fig. 5.7(d) for NF-ASM simulation and in 5.8(d) for FL-ASM simulation.

FL-ASM takes slightly more amount of time to complete the tracking as compared to NF-ASM. Similarly, MATLAB's QP algorithm is approximately 1.5 times slower to solve the trajectory tracking problem. Finally, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.7(b) and 5.8(b) but it is higher for MATLAB's QP algorithm as seen in Fig. 5.9(b).

Figs. 5.7(e-h), 5.8(e-h) and 5.9(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and QP algorithm, respectively. There is relatively large error magnitude at the beginning of the iterations but the error reduces significantly in the later parts of the iterations

Table 5.3: Circular Trajectory Tracking Results under the Constraint Set $Cc3$

Algorithm	Constraints				Total Cost ($J \times 10^{-8}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}	151	20572	14.1
FL-ASM	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}	151	24056	15
MATLAB's QP	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}	151	—	22.8

which shows that all three algorithms are capable of solving the trajectory tracking problem under the constraint set $Cc3$.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.4(i-j), 5.5(i-j) and 5.6(i-j) , respectively. As seen in these figures, the control inputs do not exceed the constraint limits imposed in $Cc3$.

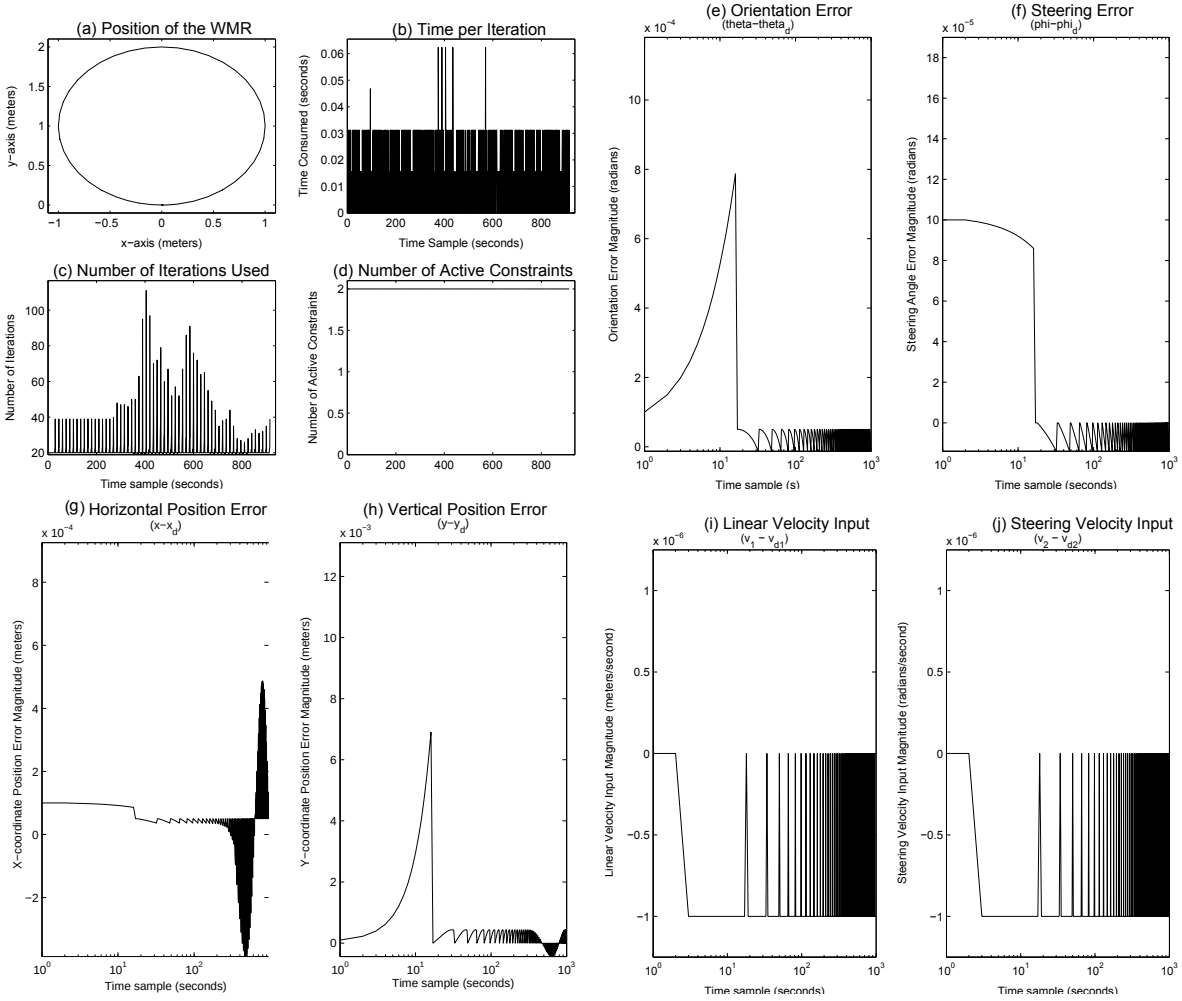

Figure 5.7: NF-ASM performance under constraint Cc3

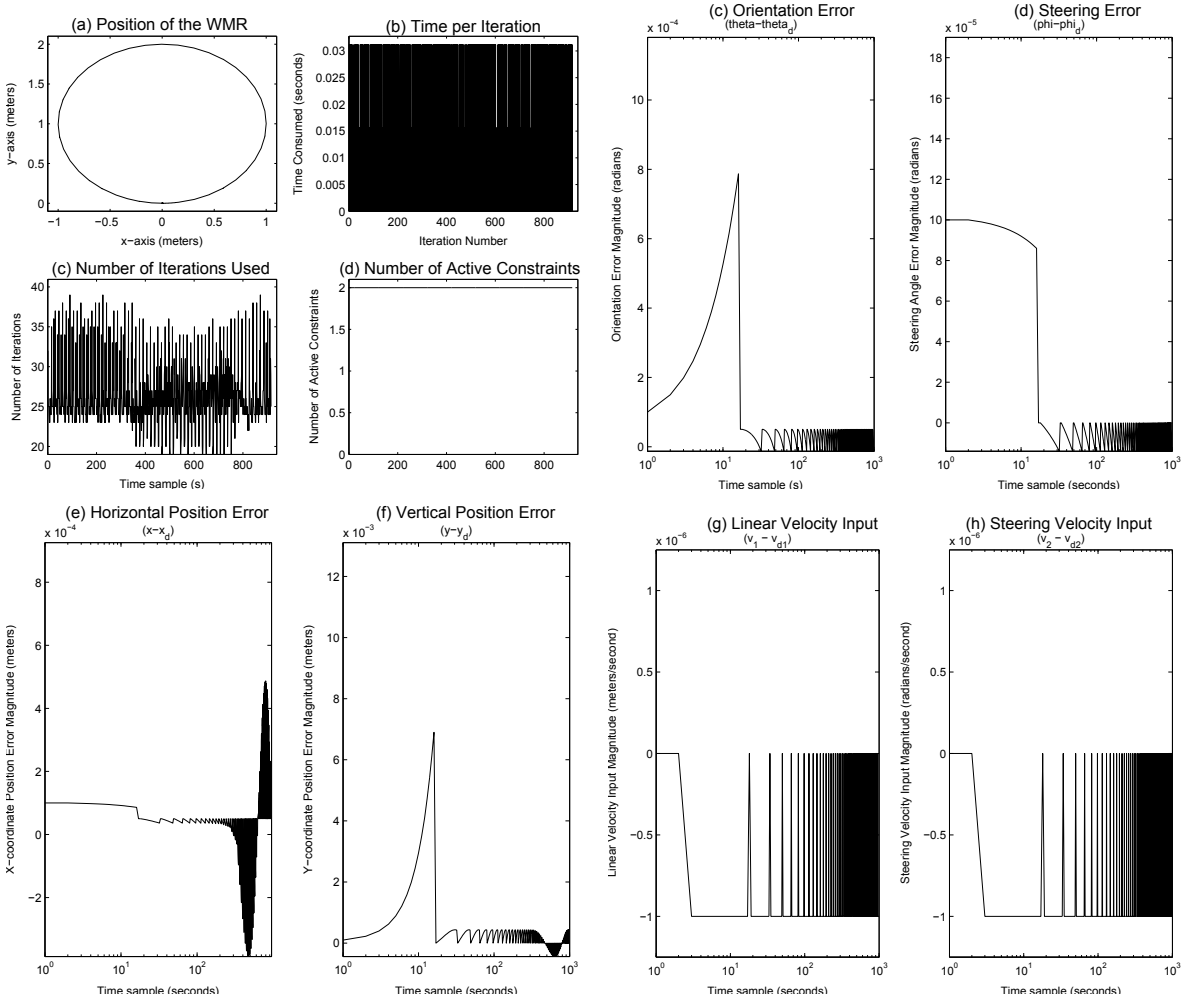

Figure 5.8: FL-ASM performance under constraint Cc3

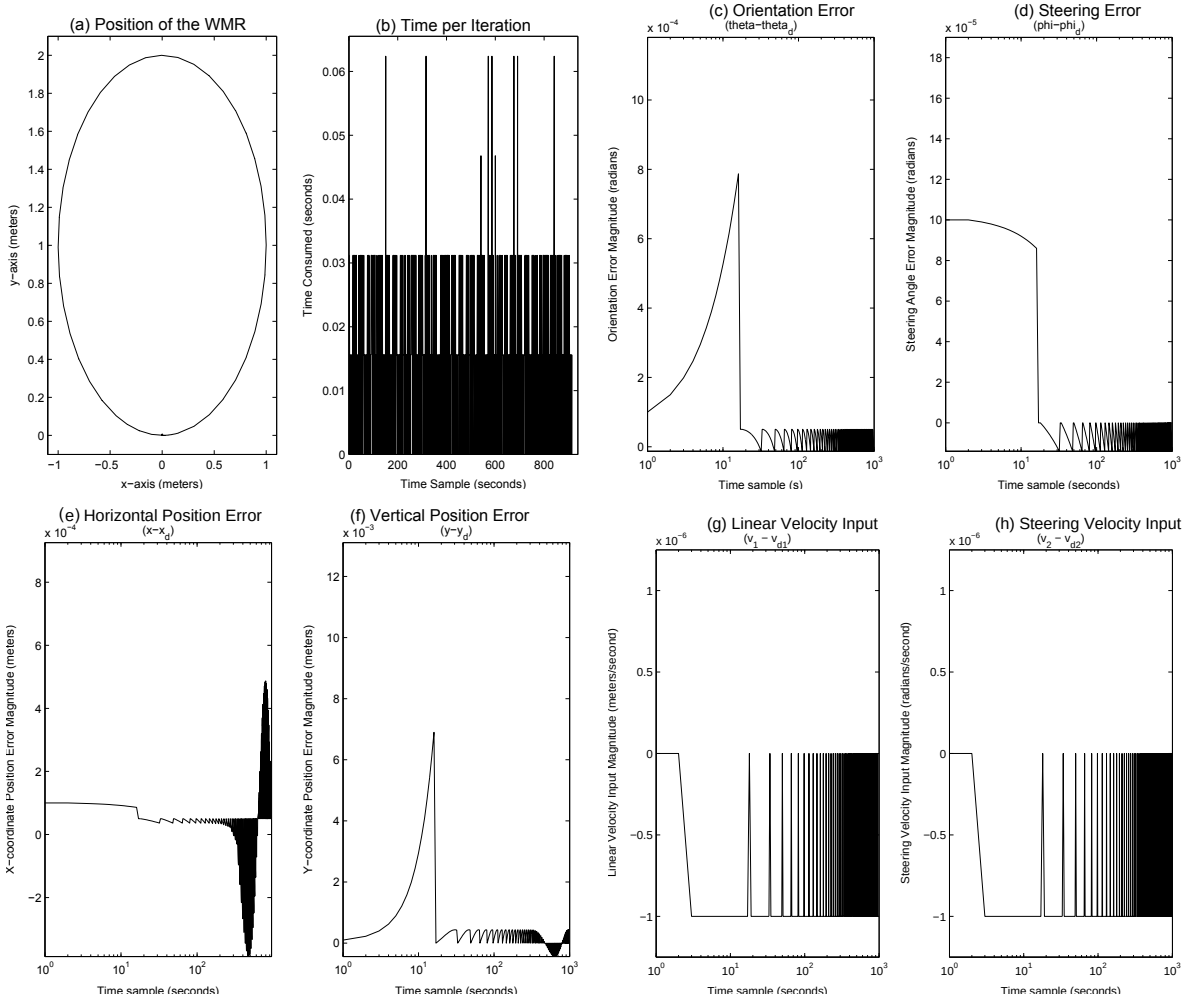

Figure 5.9: MATLAB's QP performance under constraint Cc3

5.1.4 Summary of Results

In this section, we provide a brief summary of results obtained in the previous sections for circular trajectory tracking. We also make a comparison between NF-ASM and the other two algorithms in terms of the number of iterations used and the total time consumed for simulations. A general trend in the simulation results for all the three algorithms can be observed; as we tighten the constraints on the inputs, constraints stay active for a longer period leading to an increased number of iterations used which, in turn, increases the overall simulation time.

Since the time per iteration under constraint set Cc1 (shown in Figs. 5.1 (b), 5.2 (b) and 5.3 (b)), Cc2 (shown in Figs. 5.4 (b), 5.5 (b) and 5.6 (b)) and Cc3 (shown in Figs. 5.7 (b), 5.8 (b) and 5.9 (b)) is much lower than the sampling time of $0.1\ seconds$ in all simulation cases, real-time control is achieved with all the three algorithms.

As seen in Figs. 5.1(i-h), 5.4(i-h) and 5.7(i-h) for NF-ASM, Figs. 5.2(i-h), 5.5(i-h) and 5.8(i-h) for FL-ASM and Figs. 5.3(i-h), 5.6(i-h) and 5.9(i-h) for MATLAB's QP algorithm, the state errors do not stay at a zero value. They increase in a periodic manner. This happens because the model of the WMR is time-varying which leads to a change in dynamics of the WMR. This change in system dynamics with time results in a need for the re-execution of MPC. With each change in the dynamics of the WMR model, the application of MPC algorithm re-stabilizes the system causing the error to reach a zero value as seen in the figures. It shows that all three algorithms are capable of driving the error model of the linearized time-varying WMR to stability regardless of the magnitude of constraints applied to the inputs.

5.1.4.1 NF-ASM vs FL-ASM

Table 5.4 reiterates the trajectory tracking results of NF-ASM and FL-ASM for comparison purposes. NF-ASM gives better performance in terms of number of iterations used and the total time consumed when the constraints are active at all times during the simulation, i.e. when $Cc3$ is used. FL-ASM is a better choice if the constraint are active only for a fraction of the total time of the

Table 5.4: NF-ASM vs FL-ASM for Circular Trajectory Tracking

Algorithm	Constraints				Total Cost ($J \times 10^{-8}$)	Number of Iterations	Total Time Consumed
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
FL-ASM	-1	-1	1	1	1.73	0	1.26
NF-ASM	-1	-1	1	1	1.73	0	1.26
FL-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	8.91	5577	6.36
NF-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	8.91	7545	7.25
FL-ASM	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}	151	24056	15
NF-ASM	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}	151	20572	14.1

simulation as in the case when $Cc2$ is used.

Constraints become active when the magnitude of the inputs required to achieve the desired set-point is greater than the magnitude of the limits imposed on the inputs. The inputs in this case are the linear velocity error and the steering velocity error. Therefore, the constraints on the inputs will become active when the linear and steering velocities change by a magnitude that exceeds the limit imposed by the constraints. From a practical perspective, it means that the constraints have a higher chance of staying active for a longer period during the trajectory if the speed and the steering velocity keep on changing. Hence, if there are a lot of turns and a lot of obstacles (or traffic) around the WMR (or a vehicle), NF-ASM would be a better option for trajectory tracking as it gives better results in simulation set $Cc3$ in which the constraints stay active for a longer period.

5.1.4.2 NF-ASM vs MATLAB's MATLAB's QP algorithm

Table 5.5 reiterates the trajectory tracking results of NF-ASM and MATLAB's QP algorithm for comparison purposes. NF-ASM gives better performance in terms of total time consumed in all three simulation cases.

Table 5.5: NF-ASM vs MATLAB's QP algorithm for Circular Trajectory Tracking

Algorithm	Constraints				Total Cost ($J \times 10^{-8}$)	Total Time Consumed
	lower bounds		upper bounds			
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)		
MATLAB's QP	-1	-1	1	1	1.73	6.44
NF-ASM	-1	-1	1	1	1.73	1.26
MATLAB's QP	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	8.91	13.2
NF-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	8.91	7.25
MATLAB's QP	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}	151	22.8
NF-ASM	-10^{-6}	-10^{-6}	10^{-6}	10^{-6}	151	14.1

5.2 Square Trajectory Tracking

In this section, simulations are conducted to track a square trajectory with smooth corners using NF-ASM. Smooth corners in the square trajectory are used because of the non-holonomy of the WMR model, i.e. it's inability to achieve certain maneuvers instantaneously. In this case, we use the smooth corners to avoid right-angled turns at the corners of the square trajectory. The generated reference trajectory is given in Fig. 4.5. We apply NF-ASM to the WMR using five different sets of constraints $Cs1 - Cs5$. The performance of NF-ASM is compared to the performance of FL-ASM and QP algorithm using the performance metrics explained in the previous Chapter and the results are presented and discussed in the following subsections.

5.2.1 Simulations under Constraint Set Cs1

We start with relaxed constraints on the inputs, i.e. Cs1: $-1\ m/s \leq v_1 - v_{d1} \leq 1\ m/s$ and $-1\ rad/s \leq v_2 - v_{d2} \leq 1\ rad/s$. NF-ASM demonstrates good results in terms of trajectory tracking performance as shown in Fig. 5.10(a). None of the constraints become active during the simulation, as shown in Fig. 5.10(d), because the constraints on each input are too relaxed. Hence, unconstrained control optimization is performed and no active set algorithm is required to solve

the tracking problem, leading to zero iterations used as shown in Fig. 5.10(c). The state errors are shown in Figs. 5.10(e-h) and the control inputs are shown in Figs. 5.10(i-j).

We also apply FL-ASM and MATLAB's QP algorithm to the WMR using the same simulation setup and the constraint set $Cs1$. The results of these simulations are shown in Figs. 5.11 and 5.12. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.11(a) and 5.12(a).

In FL-ASM simulation, none of the constraints become active during the simulation, as shown in Fig. 5.11(d), because the constraints on each input are too relaxed as in the case of NF-ASM simulation. Hence, unconstrained control optimization is performed and no active set algorithm is required to solve the tracking problem, leading to zero iterations used as shown in Fig. 5.11(c).

Table 5.6 summarizes our results for simulations under the constraint set $Cs1$. All algorithms give the same cost function value (i.e. $J = 3.47 \times 10^{-8}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased comparison of their performance in terms of computational workload and simulation time. NF-ASM and FL-ASM algorithms take approximately same amount of simulation time because both algorithms are performing unconstrained optimization which is exactly the same for both algorithm. MATLAB's QP algorithm, on the other hand, is approximately $twice$ as slow to solve the trajectory tracking problem. Also, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.10(b) and 5.11(b) but it is higher for Quadratic Programming algorithm as seen in Fig. 5.12(b).

Figs. 5.10(e-h), 5.11(e-h) and 5.12(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and QP algorithm, respectively. There is relatively large error magnitude at the beginning of the iterations but the error reduces significantly in the later parts of the iterations which shows that all three algorithms are capable of solving the trajectory tracking problem. Note, however, that the error increasing again in some cases. This is because the WMR model is a time-varying. The changes in the dynamics of the system increases the error again and the MPC algorithm

Table 5.6: Square Trajectory Tracking Results under the Constraint Set $Cs1$

Algorithm	Constraints				Total Cost ($J \times 10^{-8}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-1	-1	1	1	3.47	0	8.47
FL-ASM	-1	-1	1	1	3.47	0	10
MATLAB's QP	-1	-1	1	1	3.47	—	17.8

adjusts to re-stabilize the system, i.e. making its states go to zero again.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.10(i-j), 5.11(i-j) and 5.12(i-j) , respectively. Since, the control inputs do not exceed the constraint limits imposed in $Cs1$, unconstrained control optimization is performed by all three algorithms.

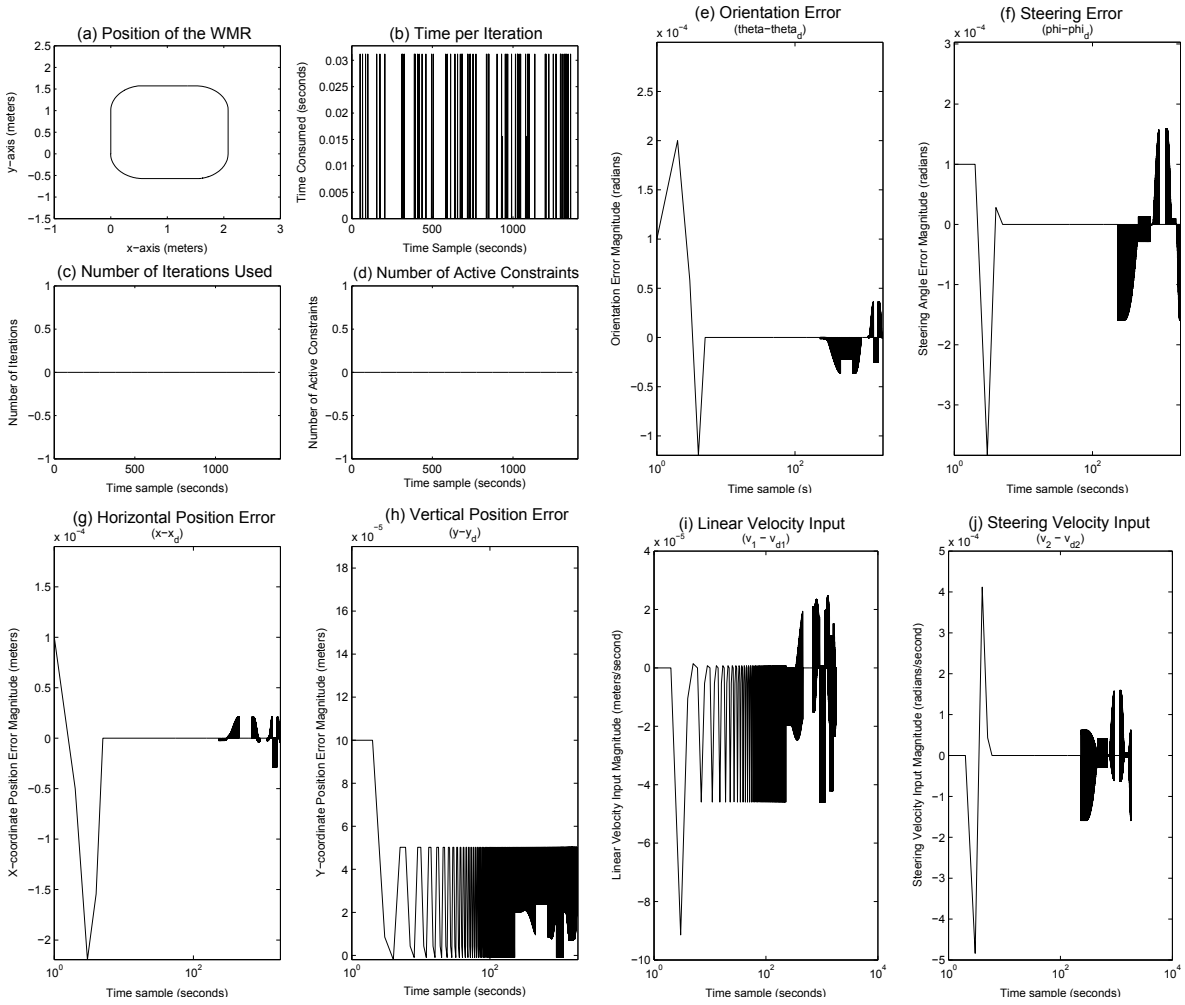

Figure 5.10: NF-ASM performance under constraint Cs1

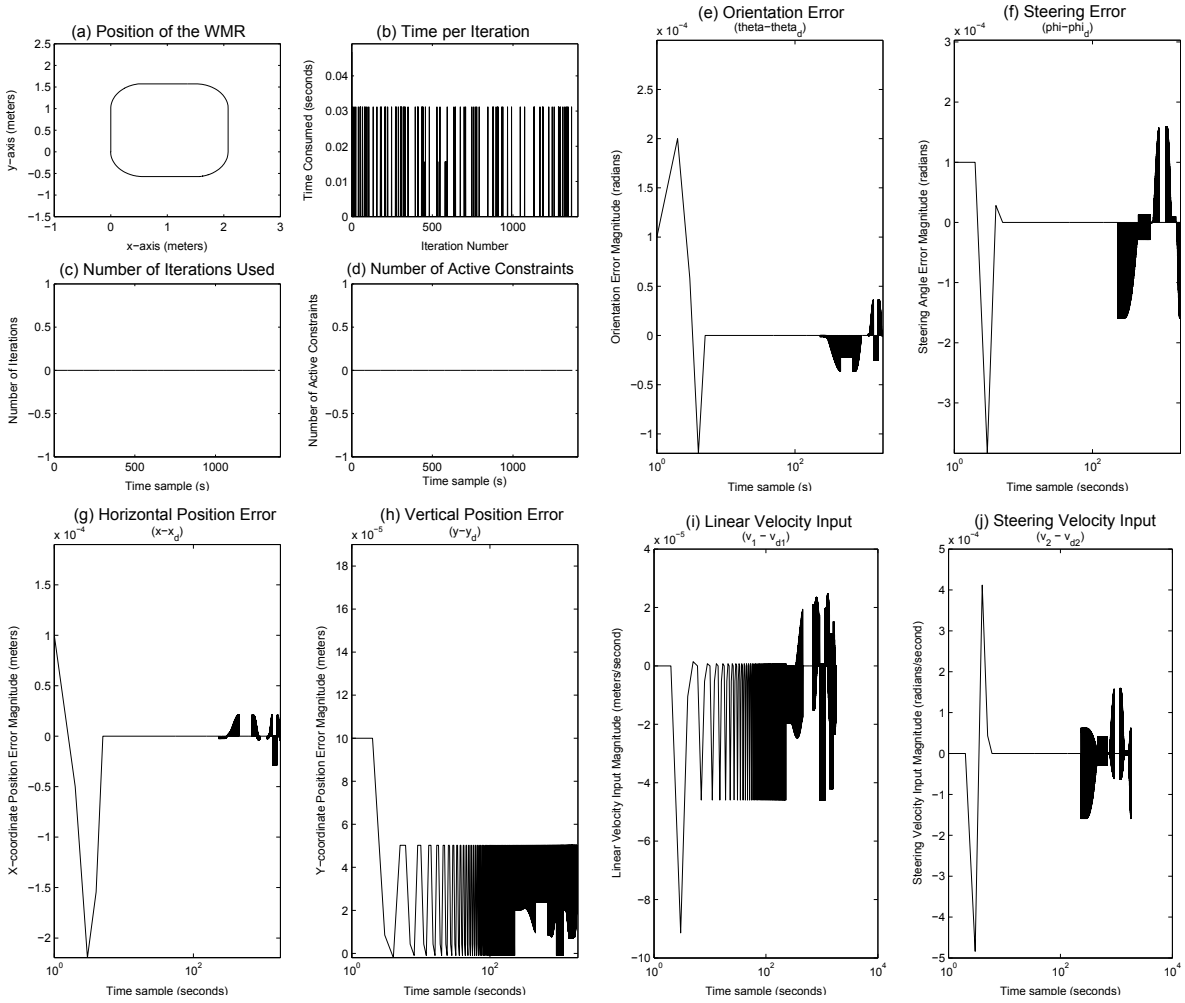

Figure 5.11: FL-ASM performance under constraint Cs1

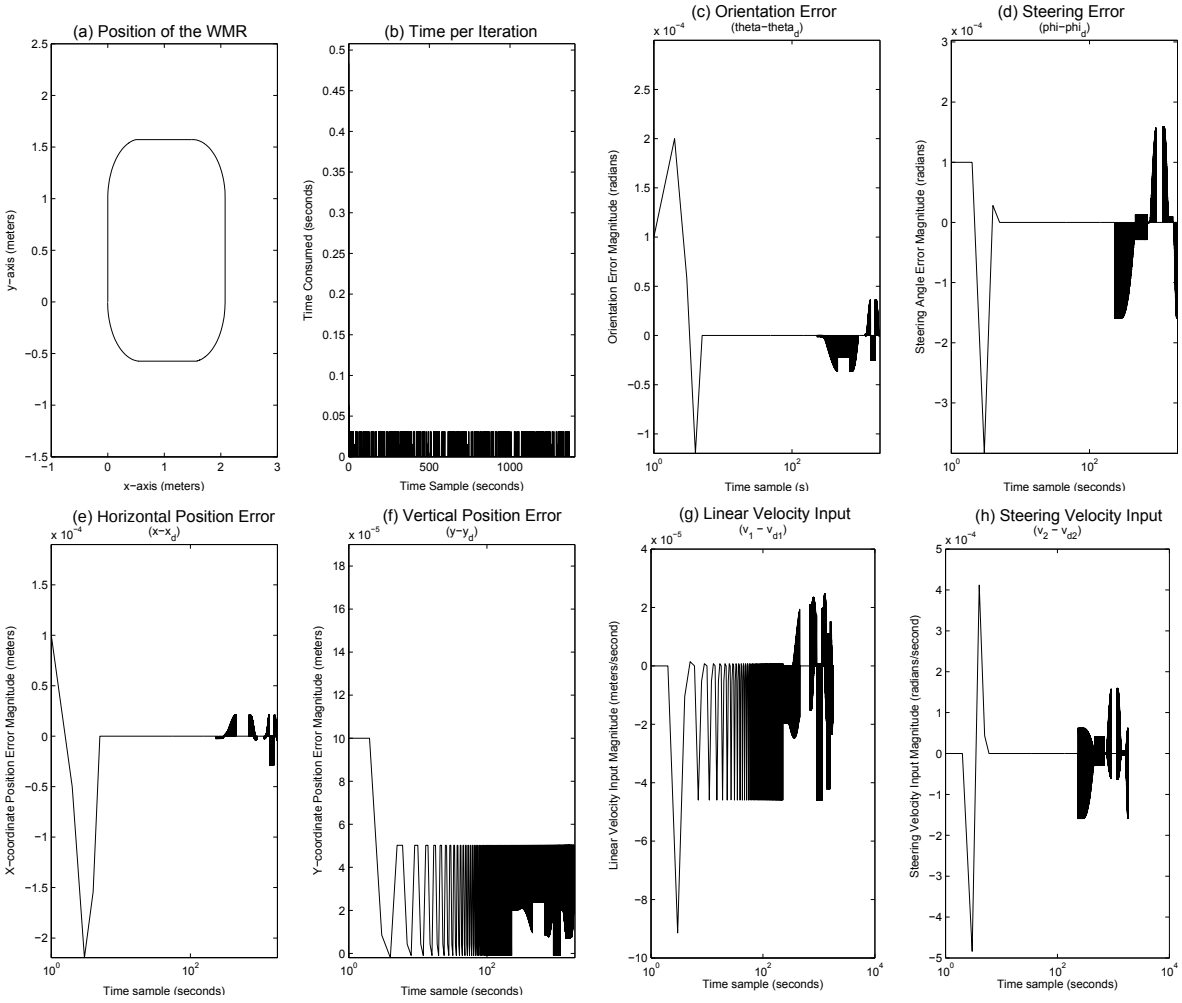

Figure 5.12: MATLAB's QP performance under constraint Cs1

5.2.2 Simulations under Constraint Set Cs2

We tighten our constraints to make them active in our second simulation, i.e. Cs2: $-1\times10^{-4}\ m/s \leq v_1 - v_{d1} \leq 1 \times 10^{-4}\ m/s$ and $-1 \times 10^{-4}\ rad/s \leq v_2 - v_{d2} \leq 1 \times 10^{-4}\ rad/s$. The tracking results are shown in Fig. 5.13(a) along with the state errors shown in Figs. 5.13(e-h). The number of active constraints at a given point toggles between zero and one as seen in Fig. 5.13(d). The number of iterations used at each time sample are shown in Fig. 5.13(c). The control inputs to the linearized WMR model are shown in Figs. 5.13(i-j). It is clear from the figures that the linear velocity input never hits the constraints during the simulation. The steering velocity input in Fig. 5.13(j) becomes active as soon as the input saturates. Hence, for square trajectory under the constraint set $Cs2$, only the constraint on steering velocity becomes active. It is important to note that the constraint on steering velocity only becomes active when the WMR is turning as the steering velocity does not change for straight paths. Hence, the smooth corners in the square are the parts of the trajectory where the steering constraint would become active.

We also apply FL-ASM and MATLAB's QP algorithm to the WMR using the same simulation setup and the constraint set $Cs2$. The results of these simulations are shown in Figs. 5.14 and 5.15. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.14(a) and 5.15(a).

Table 5.7 summarizes our results for simulations under the constraint set $Cs2$. All algorithms give the same cost function value (i.e. $J = 1.57 \times 10^{-8}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased comparison of their performance in terms of computational workload and simulation time. NF-ASM takes less amount of time to complete the tracking as compared to FL-ASM. MATLAB's QP algorithm is also approximately two times slower to solve the trajectory tracking problem. Also, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.13(b) and 5.14(b) but it is higher for MATLAB's QP algorithm as seen in Fig. 5.15(b).

The number of iterations required to solve the trajectory tracking problem using FL-ASM is

Table 5.7: Square Trajectory Tracking Results under the Constraint Set $Cs2$

Algorithm	Constraints				Total Cost ($J \times 10^{-8}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}	1.57	421	11.9
FL-ASM	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}	1.57	278	8.5
MATLAB's QP	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}	1.57	—	17.8

slightly smaller than the number of iterations required using NF-ASM as shown in Table 5.7. Note that the constraints on the inputs are active for a very small amount of time with the constraint set $Cs2$ applied to the WMR as shown in Figs. 5.13(d) and 5.14(d).

Figs. 5.13(e-h), 5.14(e-h) and 5.15(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and QP algorithm, respectively. There is relatively large error magnitude at the beginning of the iterations but the error reduces significantly in the later parts of the iterations which shows that all three algorithms are capable of solving the trajectory tracking problem under the constraint set $Cs2$.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.13(i-j), 5.14(i-j) and 5.15(i-j) , respectively. As seen in these figures, the control inputs do not exceed the constraint limits imposed in $Cs2$.

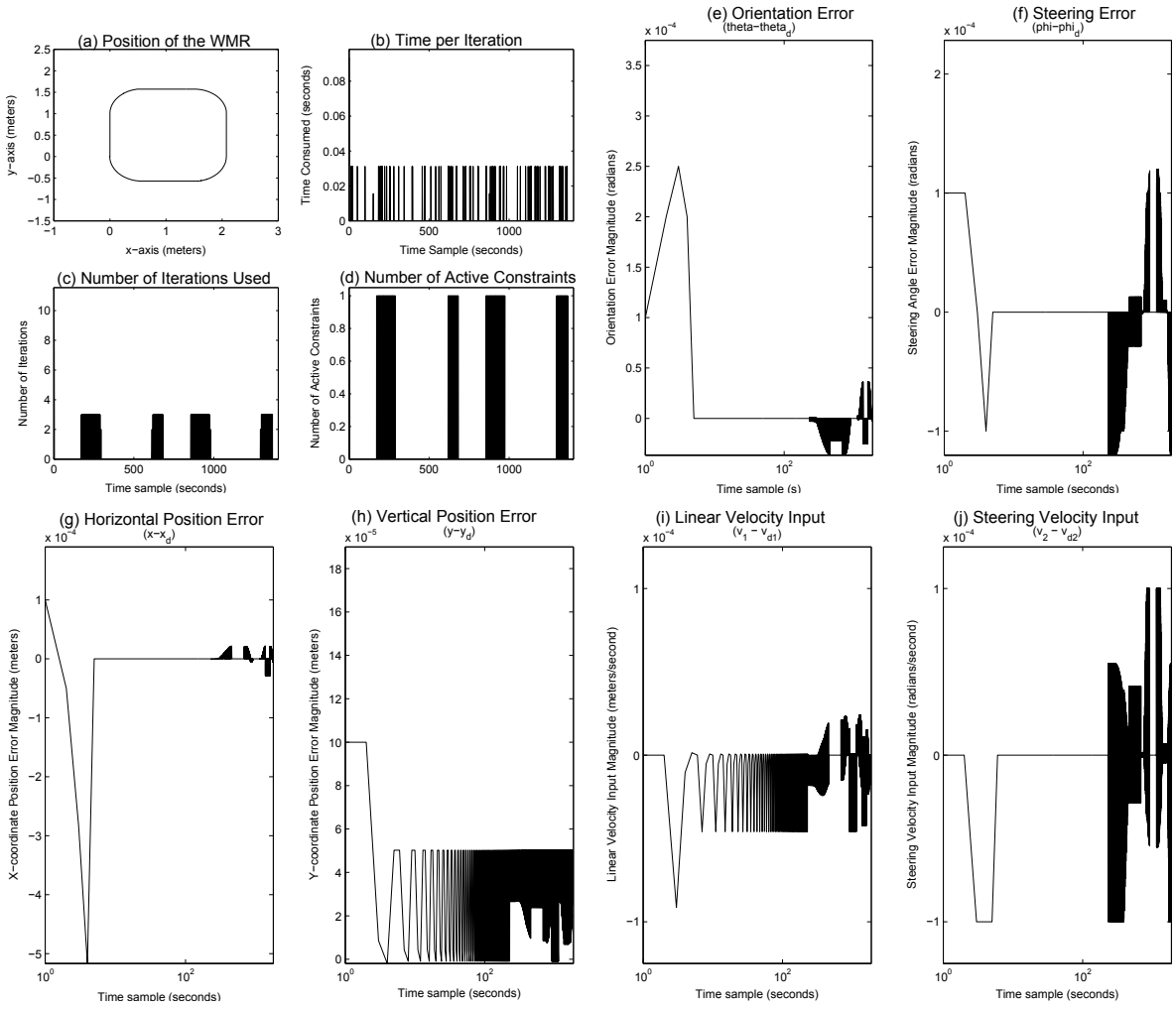

Figure 5.13: NF-ASM performance under constraint Cs2

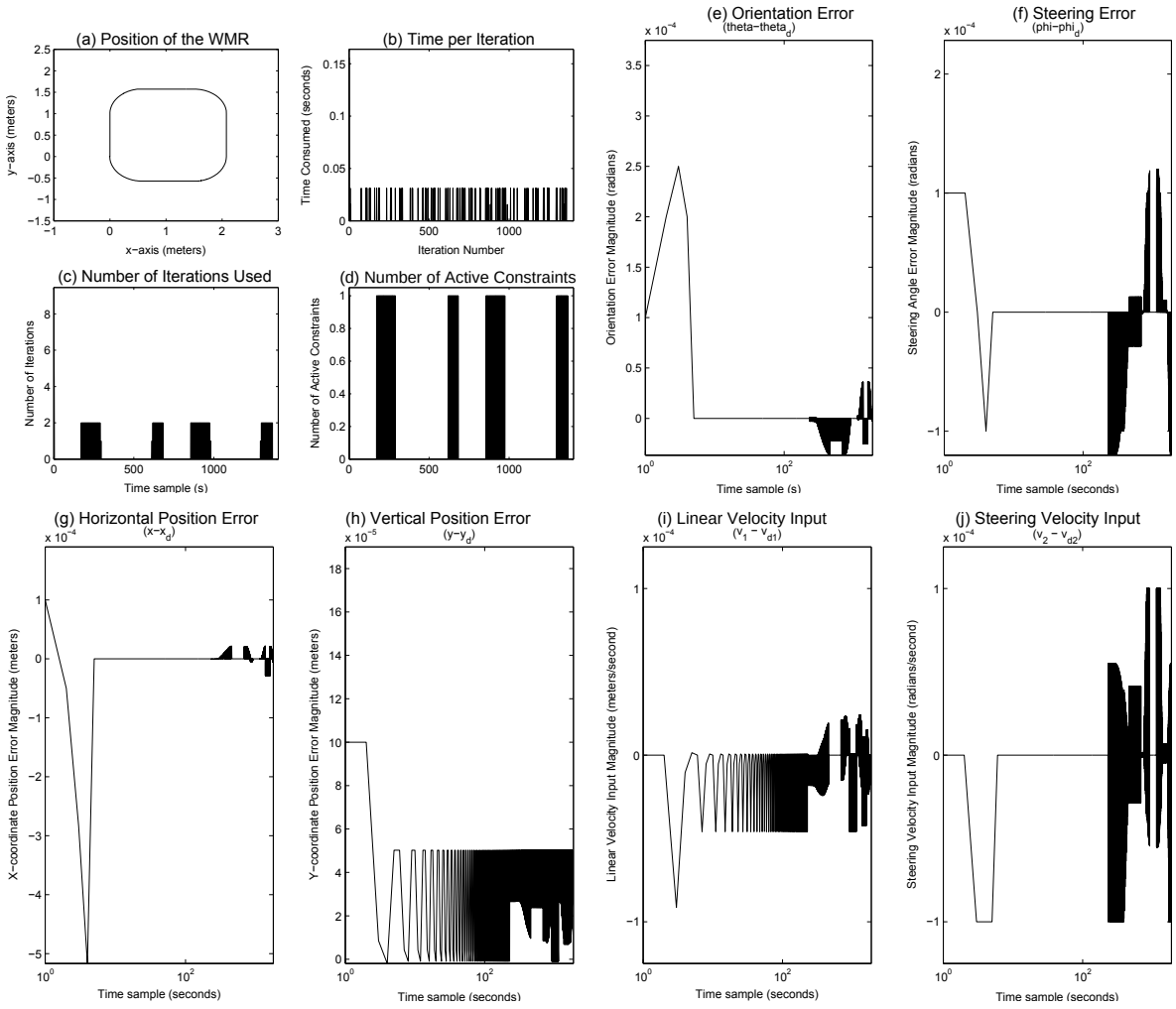

Figure 5.14: FL-ASM performance under constraint Cs2

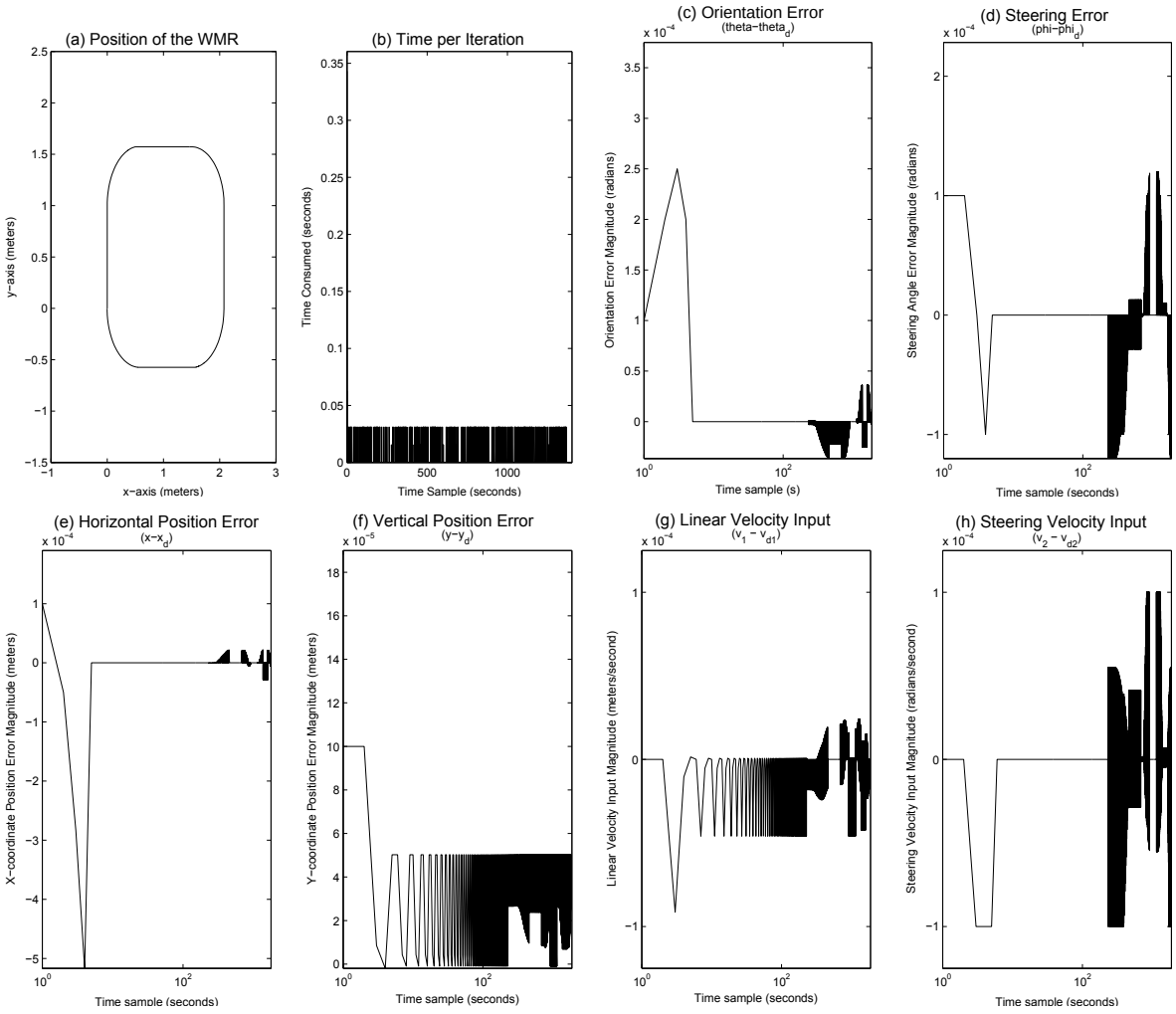

Figure 5.15: MATLAB's QP performance under constraint Cs2

5.2.3 Simulations under Constraint Set Cs3

In the third simulation, we use the constraint set Cs3: $-1 \times 10^{-5}\ m/s \leq v_1 - v_{d1} \leq 1 \times 10^{-5}\ m/s$ and $-1 \times 10^{-5}\ rad/s \leq v_2 - v_{d2} \leq 1 \times 10^{-5}\ rad/s$. The tracking results are shown in Fig. 5.16(a) along with the state errors shown in Figs. 5.16(e-h). Both constraints become active for some time during the simulation as seen in Fig. 5.16(d). The number of iterations used at each time sample are shown in Fig. 5.16(c). The control inputs to the linearized WMR model are shown in Figs. 5.16(i-j). It is clear from the figures that the control input stays within the constraints $Cs3$.

As in the previous cases, we also apply FL-ASM and MATLAB's Quadratic Programming algorithm to the WMR using the same simulation setup and the constraint set. The results of these simulations are shown in Figs. 5.17 and 5.18. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.17(a) and 5.18(a).

Table 5.8 summarizes our results for simulations under the constraint set $Cs3$. All algorithms give the same cost function value (i.e. $J = 5.91 \times 10^{-9}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased comparison of their performance in terms of computational workload and simulation time. NF-ASM consumes less number of iterations as compared to FL-ASM. Note that the constraints on the inputs are active for all times during the simulation as shown in Fig. 5.16(d) for NF-ASM simulation and in Fig. 5.17(d) for FL-ASM simulation.

FL-ASM takes roughly 1.5 times the amount of time to complete the tracking as compared to NF-ASM. Similarly, MATLAB's QP algorithm is approximately 2.5 times slower to solve the trajectory tracking problem as compared to NF-ASM. Finally, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.16(b) and 5.17(b) but it is higher for QP algorithm as seen in Fig. 5.18(b).

Figs. 5.16(e-h), 5.17(e-h) and 5.18(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and QP algorithm, respectively. There is relatively large error magnitude at the beginning of the iterations but the error reduces significantly in the later parts of the iterations

Table 5.8: Square Trajectory Tracking Results under the Constraint Set $Cs3$

Algorithm	Constraints				Total Cost ($J \times 10^{-9}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	5.91	11644	16.3
FL-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	5.91	7482	11.7
MATLAB's QP	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	5.91	—	25.4

which shows that all three algorithms are capable of solving the trajectory tracking problem under the constraint set $Cs3$.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.13(i-j), 5.14(i-j) and 5.15(i-j) , respectively. As seen in these figures, the control inputs do not exceed the constraint limits imposed in $Cs3$.

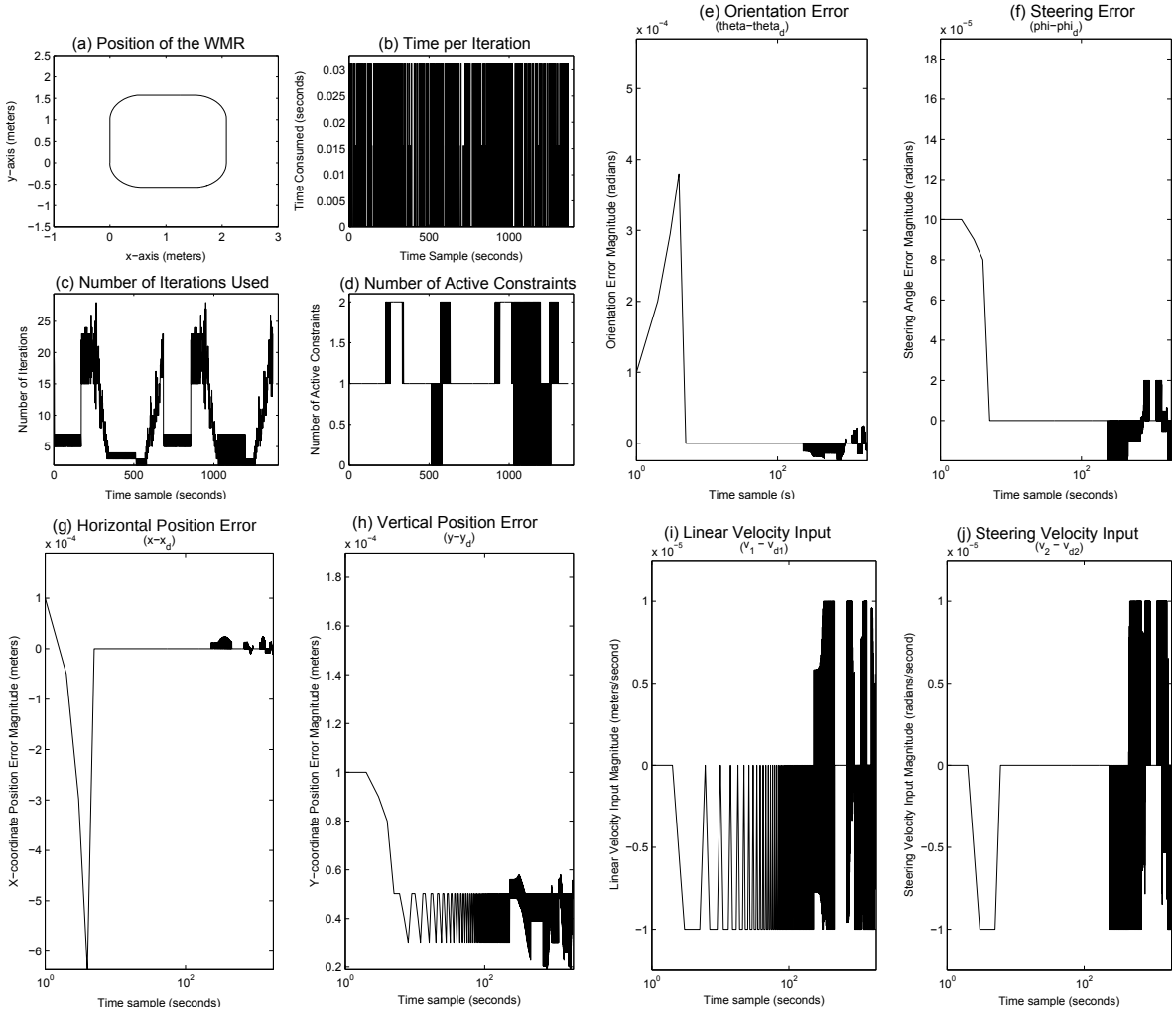

Figure 5.16: NF-ASM performance under constraint Cs3

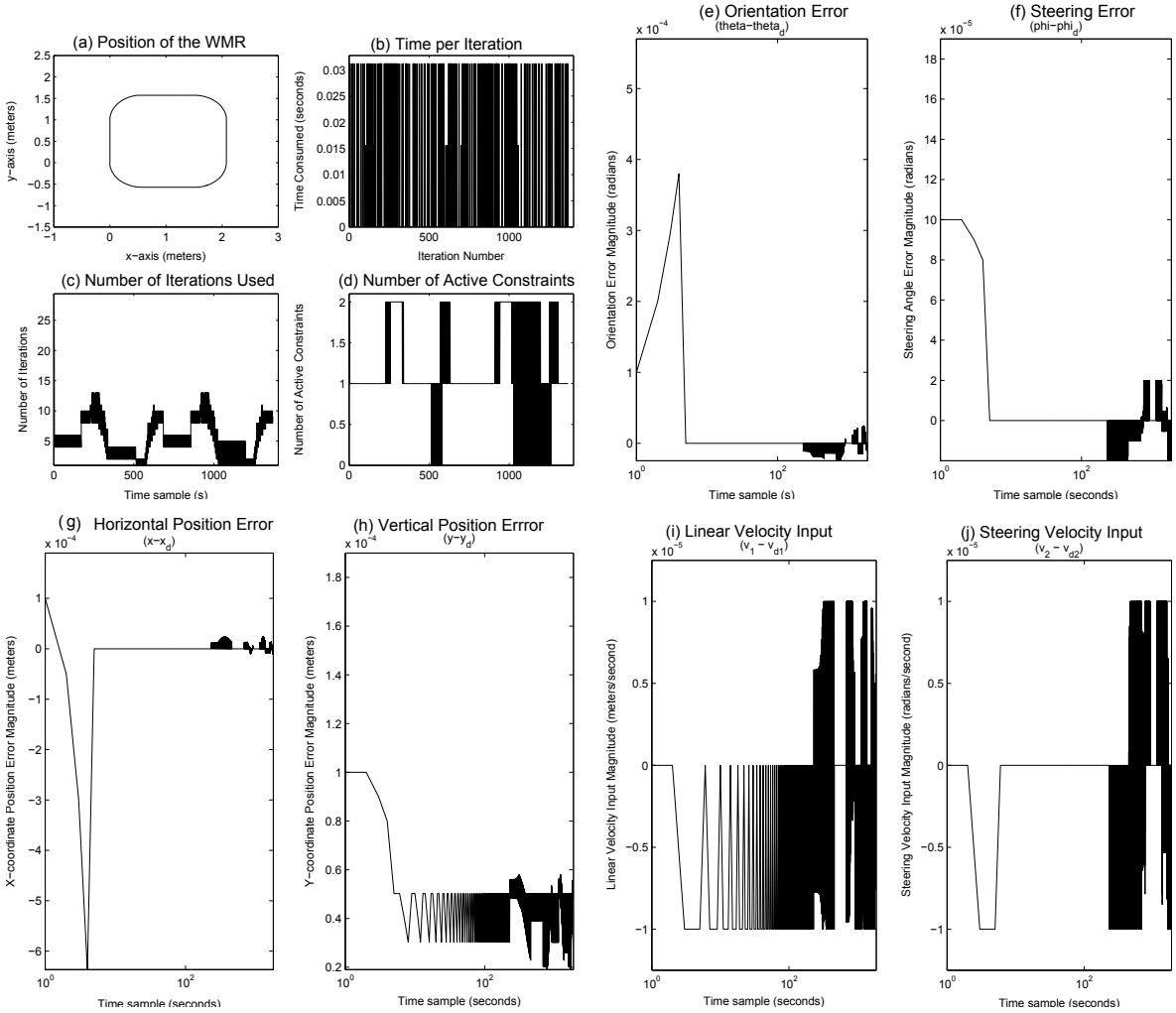

Figure 5.17: FL-ASM performance under constraint Cs3

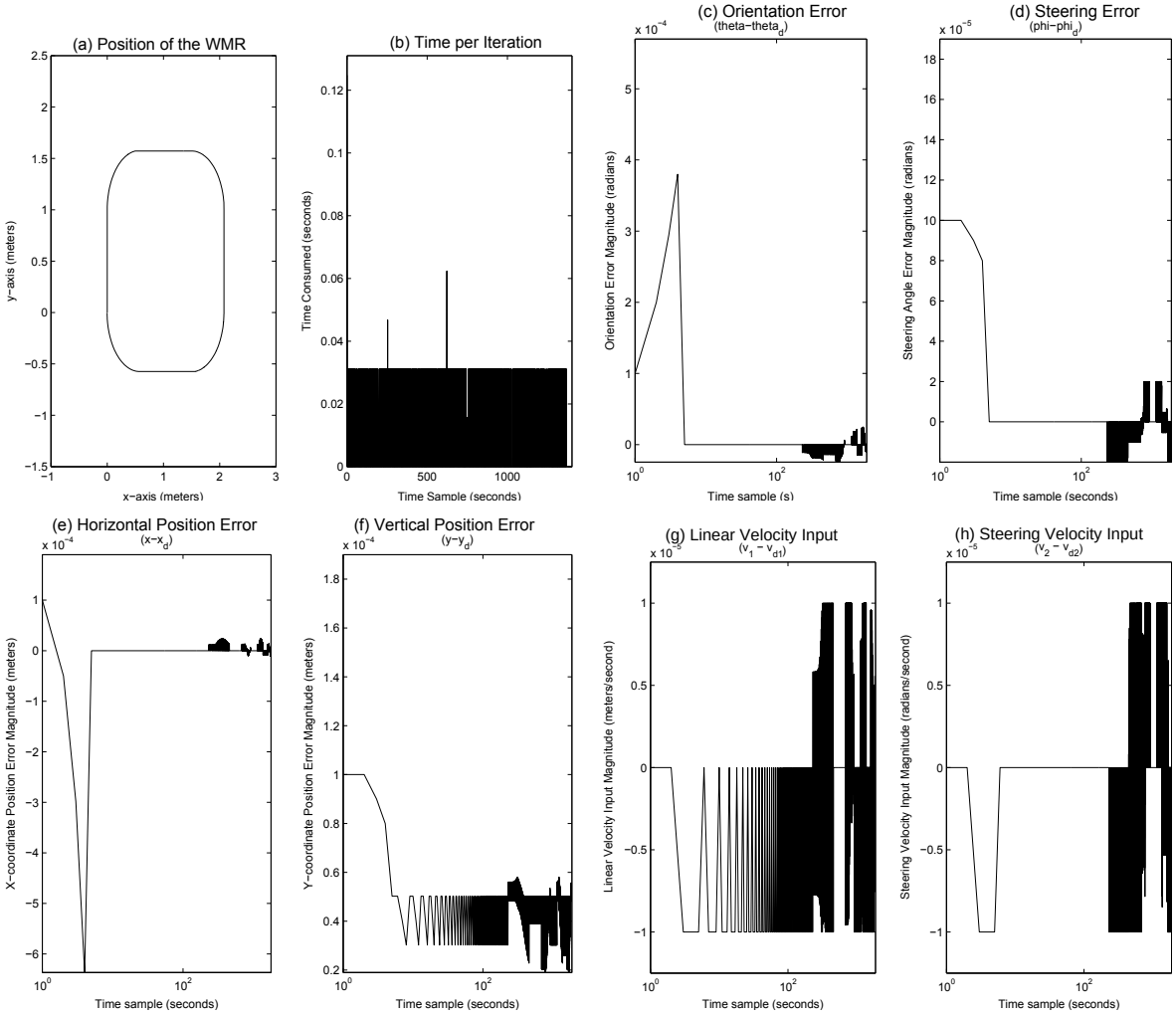

Figure 5.18: MATLAB's QP performance under constraint Cs3

5.2.4 Simulations under Constraint Set Cs4

In the fourth simulation, we use the constraint set Cs4: $-1\times10^{-8}\,m/s \leq v_1 - v_{d1} \leq 1\times10^{-8}\,m/s$ and $-1\times10^{-8}\,rad/s \leq v_2 - v_{d2} \leq 1\times10^{-8}\,rad/s$. The tracking results are shown in Fig. 5.19(a) along with the state errors shown in Figs. 5.19(e-h). Both constraints are active for a larger fraction of time seen in Fig. 5.19(d) as compared to the previous cases. The number of iterations used at each time sample are shown in Fig. 5.19(c). The control inputs to the linearized WMR model are shown in Figs. 5.19(i-j). It is clear from the figures that the control input stays within the constraints $Cs4$.

As in the previous cases, we also apply FL-ASM and MATLAB's Quadratic Programming algorithm to the WMR using the same simulation setup and the constraint set. The results of these simulations are shown in Figs. 5.20 and 5.21. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.20(a) and 5.21(a).

Table 5.9 summarizes the results for simulations under the constraint set $Cs4$. All algorithms give the same cost function value (i.e. $J = 5.046 \times 10^{-9}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased comparison of their performance in terms of computational workload and simulation time. NF-ASM consumes less number of iterations as compared to FL-ASM. Note that the constraints on the inputs are active for longer periods during the simulation as shown in Fig. 5.19(d) for NF-ASM simulation and in Fig. 5.20(d) for FL-ASM simulation.

FL-ASM takes more amount of time to complete the tracking as compared to NF-ASM. Similarly, MATLAB's QP algorithm is approximately two times slower to solve the trajectory tracking problem. Finally, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.19(b) and 5.20(b) but it is higher for Quadratic Programming algorithm as seen in Fig. 5.21(b).

Figs. 5.19(e-h), 5.20(e-h) and 5.21(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and QP algorithm, respectively. There is relatively large error magnitude

Table 5.9: Square Trajectory Tracking Results under the Constraint Set $Cs4$

Algorithm	Constraints				Total Cost ($J \times 10^{-9}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}	5.046	21351	18
FL-ASM	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}	5.046	29939	24.8
MATLAB's QP	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}	5.046	—	40

at the beginning of the iterations but the error reduces significantly in the later parts of the iterations which shows that all three algorithms are capable of solving the trajectory tracking problem under the constraint set $Cs4$.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.19(i-j), 5.20(i-j) and 5.21(i-j) , respectively. As seen in these figures, the control inputs do not exceed the constraint limits imposed in $Cs4$.

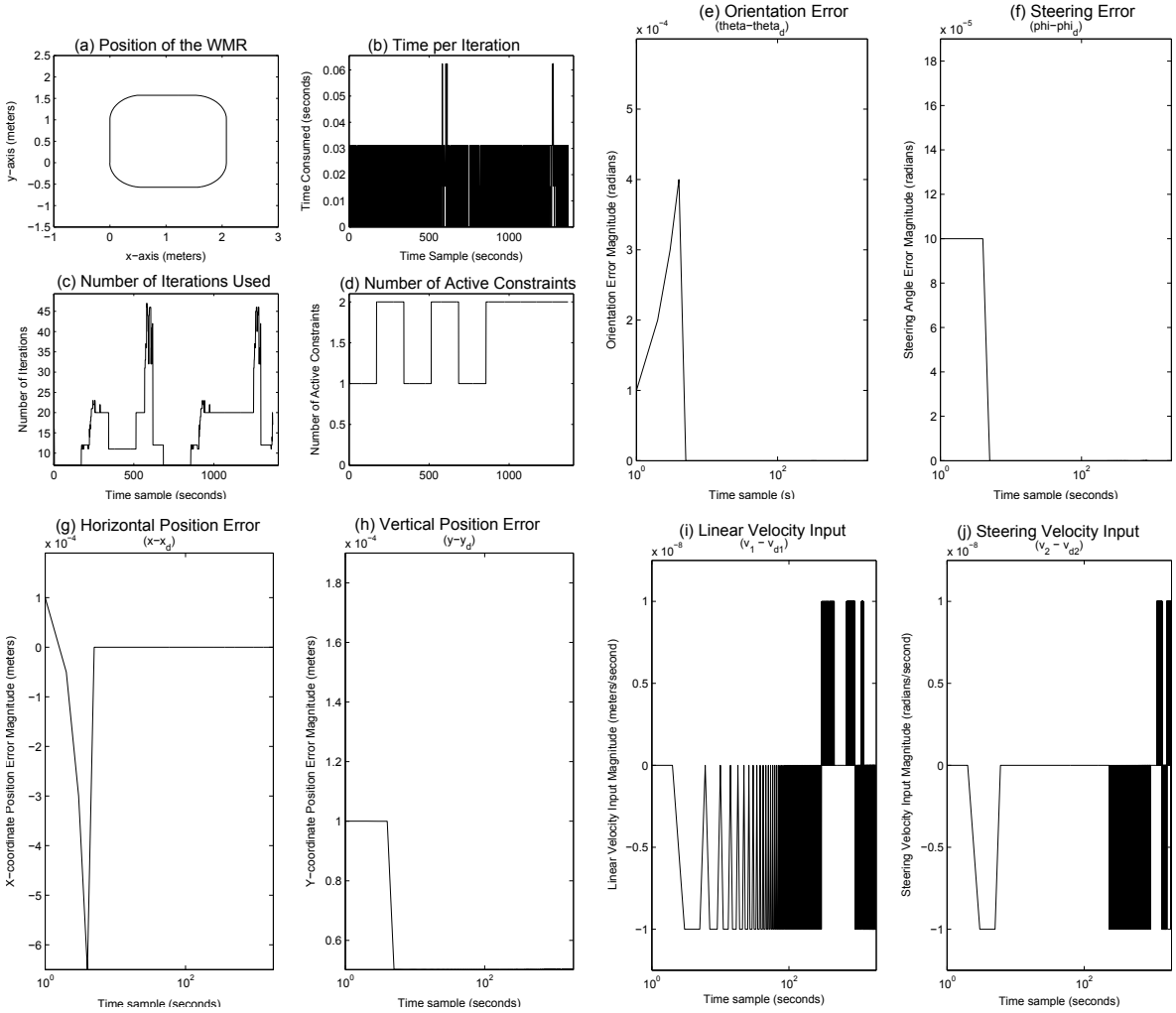

Figure 5.19: NF-ASM performance under constraint Cs4

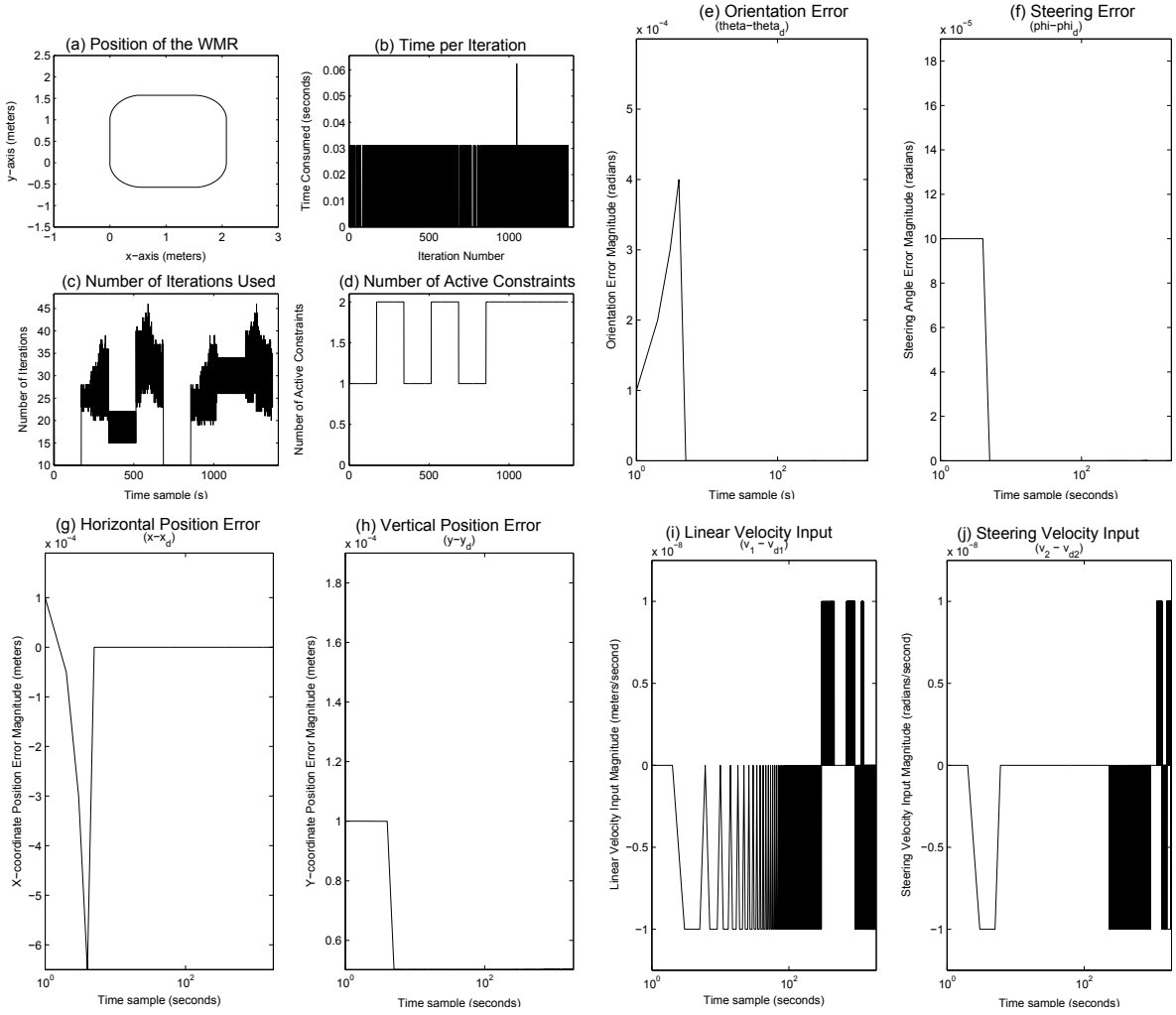

Figure 5.20: FL-ASM performance under constraint Cs4

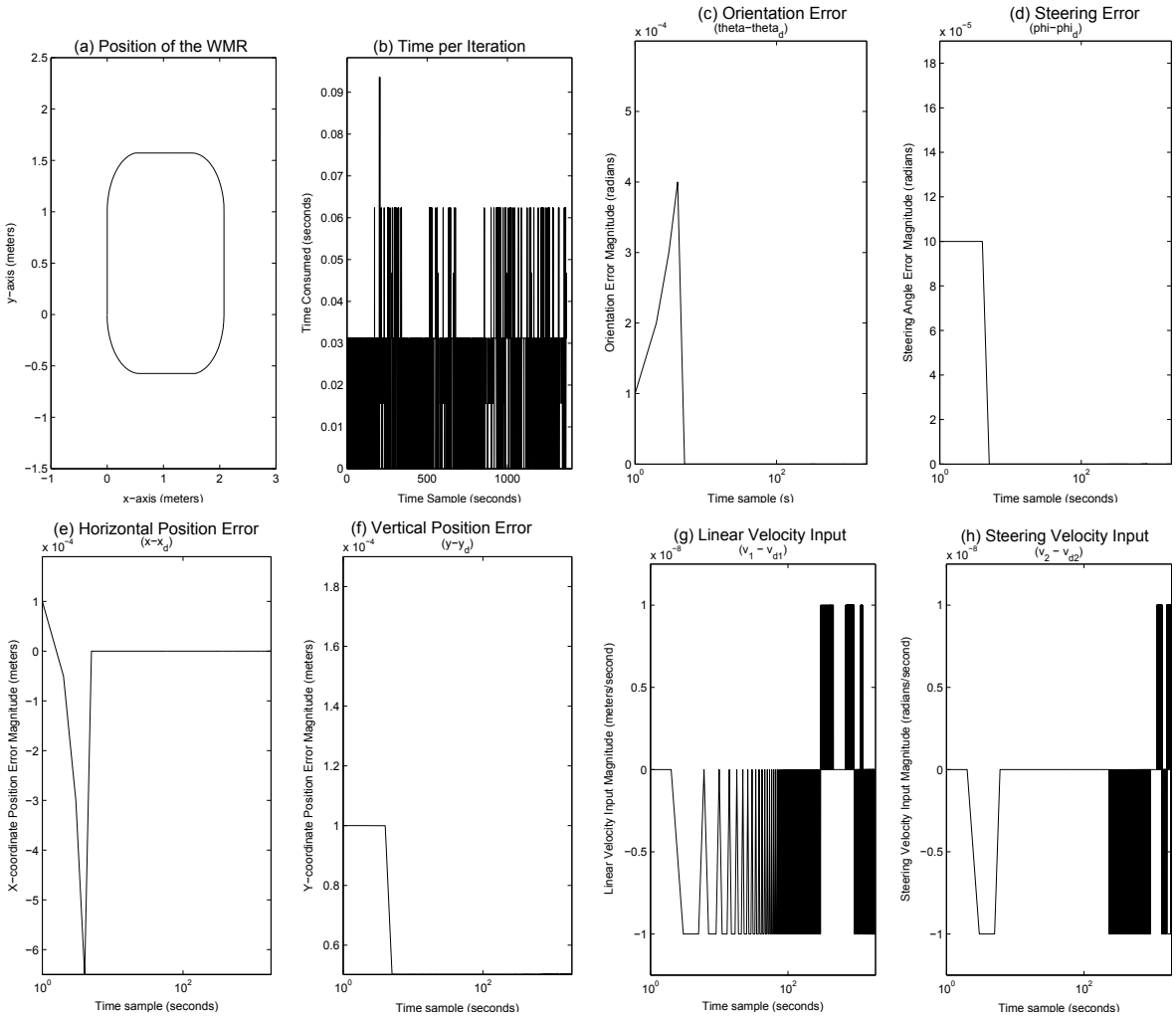

Figure 5.21: MATLAB's QP performance under constraint Cs4

5.2.5 Simulations under Constraint Set Cs5

In the third simulation, we use the constraint set Cs5: $-1 \times 10^{-9}\ m/s \leq v_1 - v_{d1} \leq 1 \times 10^{-9}\ m/s$ and $-1 \times 10^{-9}\ rad/s \leq v_2 - v_{d2} \leq 1 \times 10^{-9}\ rad/s$. The tracking results are shown in Fig. 5.22(a) along with the state errors shown in Figs. 5.22(e-h). Both constraints are active for a longer period during the simulation as seen in Fig. 5.22(d). The number of iterations used at each time sample are shown in Fig. 5.22(c). The control inputs to the linearized WMR model are shown in Figs. 5.22(i-j). It is clear from the figures that the control input stays within the constraints $Cs5$.

As in the previous cases, we also apply FL-ASM and MATLAB's Quadratic Programming algorithm to the WMR using the same simulation setup and the constraint set. The results of these simulations are shown in Figs. 5.23 and 5.24. These algorithms also demonstrate good results in terms of trajectory tracking performance as shown in Figs. 5.23(a) and 5.24(a).

Table 5.10 summarizes our results for simulations under the constraint set $Cs5$. All algorithms give the same cost function value (i.e. $J = 5.048 \times 10^{-9}$) for this simulation case which implies that all these algorithms converge to the same solution. This allows us to conduct an unbiased comparison of their performance in terms of computational workload and simulation time. NF-ASM consumes less number of iterations as compared to FL-ASM. Note that the constraints on the inputs are active for a longer period during this simulation case as shown in Fig. 5.22(d) for NF-ASM simulation and in Fig. 5.23(d) for FL-ASM simulation.

FL-ASM takes 1.5 times more amount of time to complete the tracking as compared to NF-ASM. Similarly, MATLAB's QP algorithm is approximately $twice$ as slower to solve the trajectory tracking problem as NF-ASM. Finally, the average time taken per sample is quite similar for NF-ASM and FL-ASM as seen in Figs. 5.22(b) and 5.23(b) but it is higher for MATLAB's QP algorithm as seen in Fig. 5.24(b).

Figs. 5.22(e-h), 5.23(e-h) and 5.24(e-h) show the position, orientation and steering errors using NF-ASM, FL-ASM and QP algorithm, respectively. There is relatively large error magnitude at the beginning of the iterations but the error reduces significantly in the later parts of the iterations

Table 5.10: Square Trajectory Tracking Results under the Constraint Set $Cs5$

Algorithm	Constraints				Total Cost ($J \times 10^{-9}$)	Number of Iterations	Total Time Consumed ($seconds$)
	lower bounds		upper bounds				
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)			
NF-ASM	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}	5.048	21410	18
FL-ASM	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}	5.048	31635	29.6
MATLAB's QP	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}	5.048	—	40.2

which shows that all three algorithms are capable of solving the trajectory tracking problem under the constraint set $Cs5$.

The control inputs for NF-ASM, FL-ASM and QP algorithm simulation are shown in Figs. 5.22(i-j), 5.23(i-j) and 5.24(i-j) , respectively. As seen in these figures, the control inputs do not exceed the constraint limits imposed in $Cs5$.

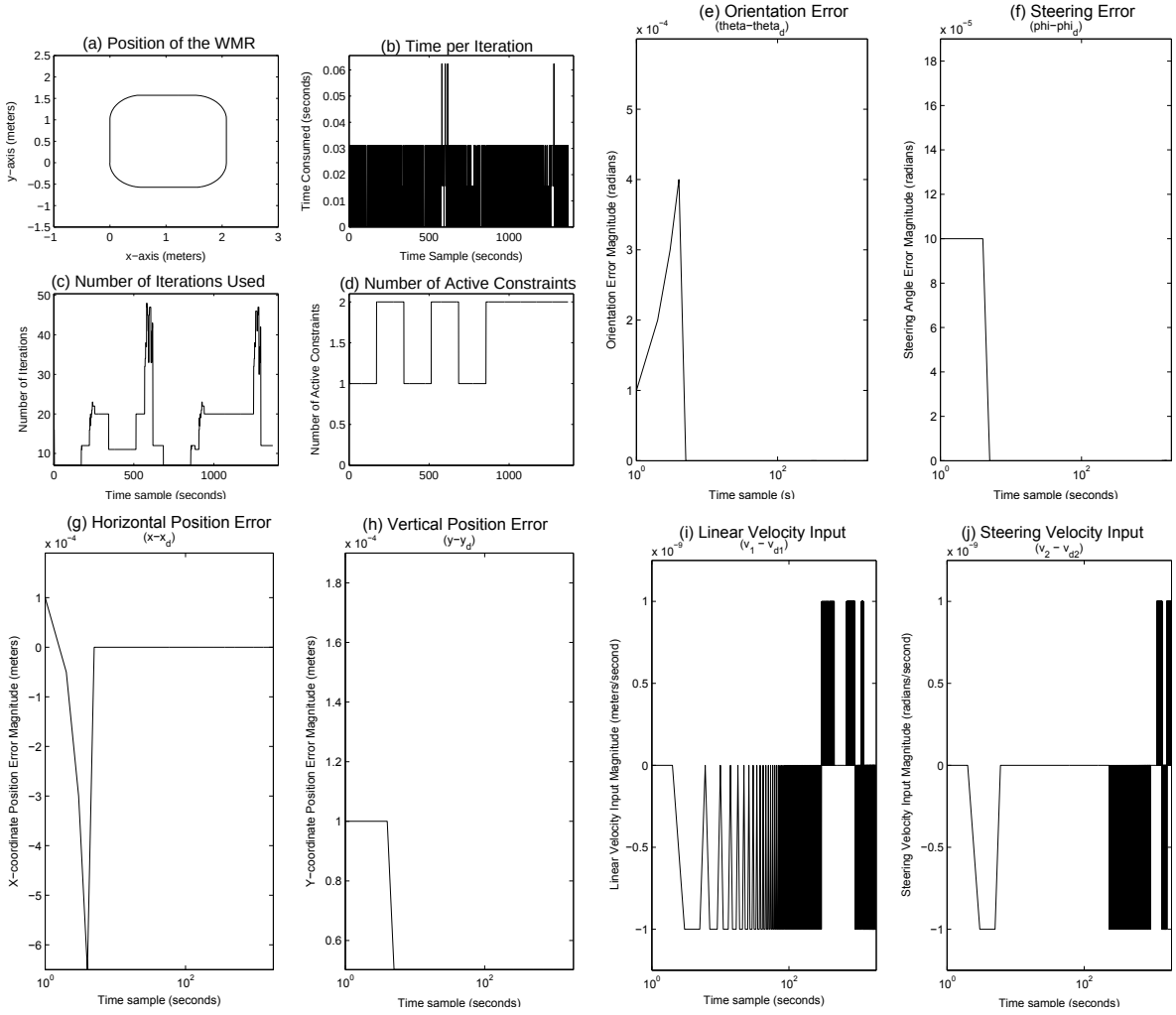

Figure 5.22: NF-ASM performance under constraint Cs5

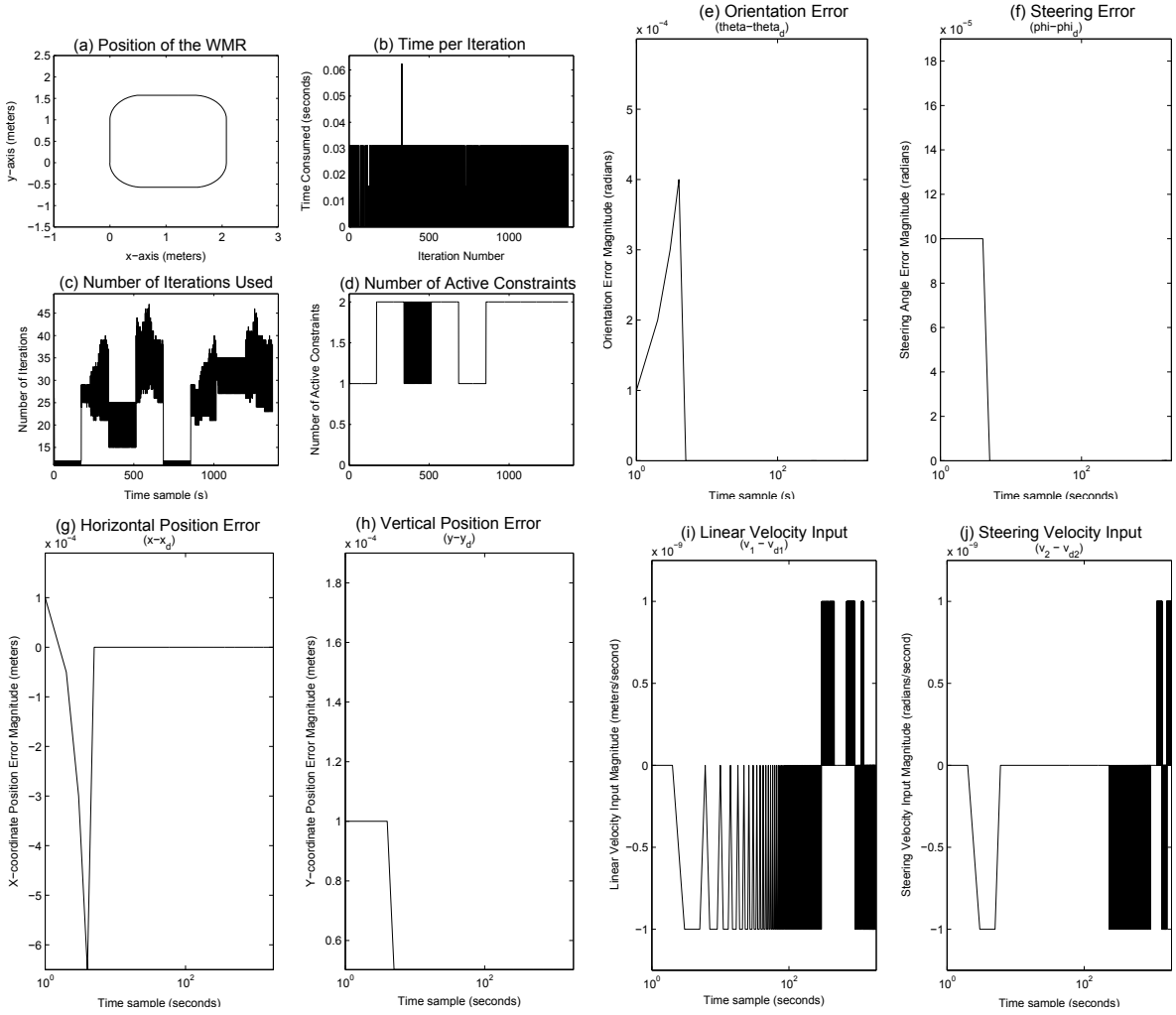

Figure 5.23: FL-ASM performance under constraint Cs5

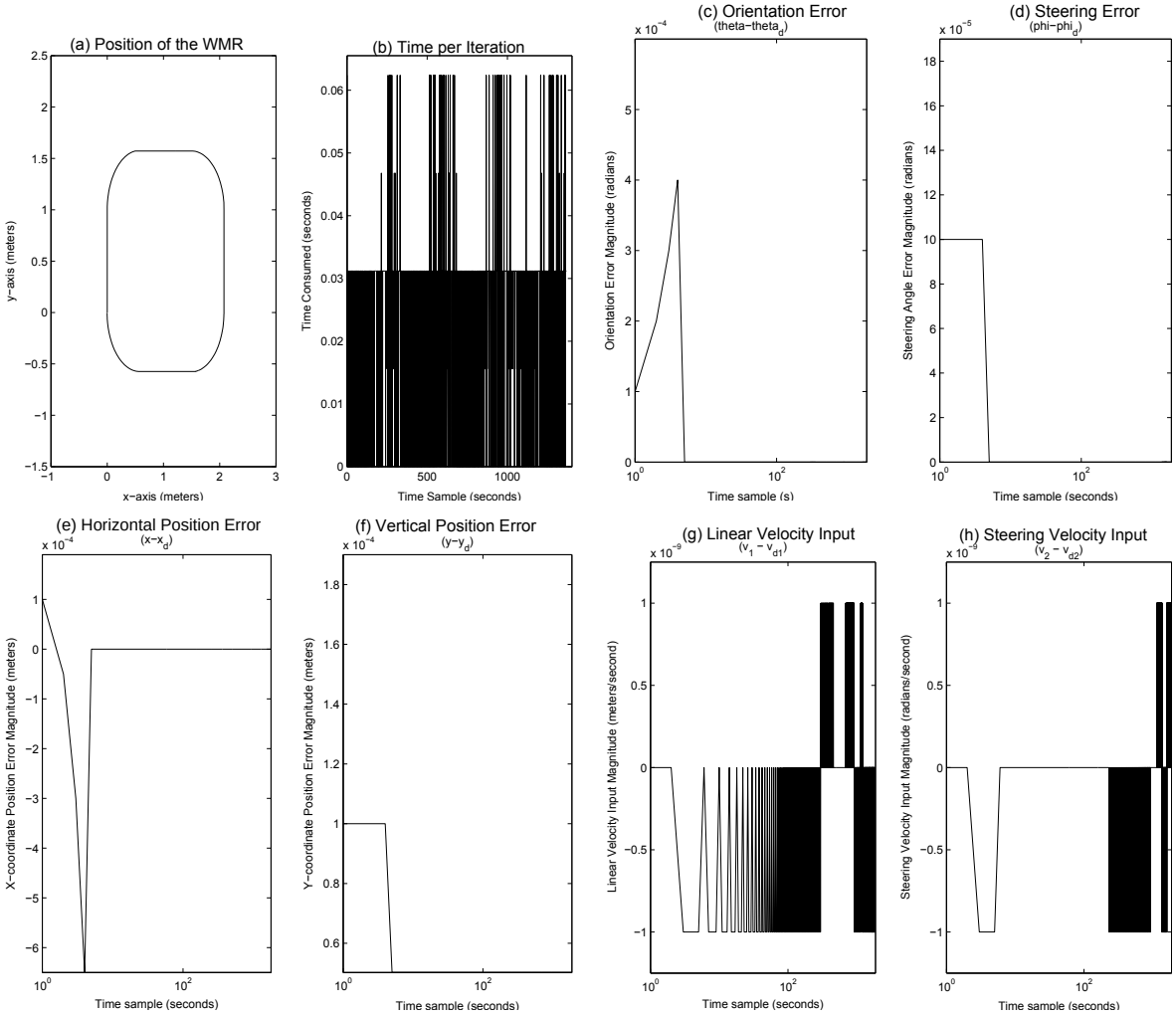

Figure 5.24: MATLAB's QP performance under constraint Cs5

5.2.6 Summary of Results

In this section, we provide a brief summary of results obtained in the previous sections for square trajectory tracking. We also make a comparison between NF-ASM and the other two algorithms in terms of the number of iterations used and the total time consumed for simulations. A general trend in the simulation results for all the three algorithms can be observed; as we tighten the constraints on the inputs, constraints stay active for a longer period leading to an increased number of iterations used which, in turn, increases the overall simulation time. Note that we observed the same trend for circular trajectory tracking.

Since the time per iteration under constraint set Cs1 (shown in Figs. 5.10 (b), 5.11 (b) and 5.12 (b)), Cs2 (shown in Figs. 5.13 (b), 5.14 (b) and 5.15 (b)), Cs3 (shown in Figs. 5.16 (b), 5.17 (b) and 5.18 (b)), Cs4 (shown in Figs. 5.19 (b), 5.20 (b) and 5.21 (b)) and Cs5 (shown in Figs. 5.22 (b), 5.23 (b) and 5.24 (b)) is much lower than the sampling time of $0.1\ seconds$ in all simulation cases, real-time control is achieved with all the three algorithms.

As seen in Figs. 5.10(i-h), 5.13(i-h), 5.16(i-h), 5.19(i-h) and 5.22(i-h) for NF-ASM, Figs. 5.11(i-h), 5.14(i-h), 5.17(i-h), 5.20(i-h) and 5.23(i-h) for FL-ASM and Figs. 5.12(i-h), 5.15(i-h), 5.18(i-h), 5.21(i-h) and 5.24(i-h) for MATLAB's quadratic programming algorithm, the state errors do not stay at a zero value. They increase in a periodic manner. This happens because the model of the WMR is time-varying which leads to a change in dynamics of the WMR. This change in system dynamics with time results in a need for the re-execution of MPC. With each change in the dynamics of the WMR model, the application of MPC algorithm re-stabilizes the system causing the error to reach a zero value as seen in the figures. It shows that all three algorithms are capable of driving the error model of the linearized time-varying WMR to stability regardless of the magnitude of constraints applied to the inputs.

5.2.6.1 NF-ASM vs FL-ASM

Table 5.11 reiterates the trajectory tracking results of NF-ASM and FL-ASM for comparison purposes. When the constraint set $Cs1$ is applied, both NF-ASM and FL-ASM demonstrate similar results in terms of trajectory tracking performance as shown in Figs. 5.10(a) and 5.11(a) . The average time taken per sample is also quite similar for both cases as seen in Figs. 5.10(b) and 5.11(b). The reason is that none of the constraints become active during the simulation, as shown in Figs. 5.10(d) and 5.11(d), because the constraints on each input are too relaxed. Hence, unconstrained control optimization is performed and no active set algorithm is required to solve the tracking problem, leading to zero iterations used as shown in Figs. 5.10(c) and 5.11(c). Therefore, both algorithms give the same cost function value $J = 3.47 \times 10^{-8}$.

We tighten the constraints and apply $Cs2$ to make them active. Again, the two algorithms give similar trajectory tracking performance as shown in Figs. 5.13(a) and 5.14(a). Both algorithms give the same cost function value $J = 1.57 \times 10^{-8}$ which shows that both algorithms are finding the same set of solutions for trajectory tracking. Also, the average time consumed per iteration is very similar for both algorithms. However, the number of iterations required to solve the trajectory tracking problem using FL-ASM is smaller than the number of iterations required using NF-ASM as shown in Table 5.11. The reason why FL-ASM performs better than NF-ASM in this case is because not all constraints stay active during the simulation as shown in Figs. 5.13(d) and 5.14(d).

We further tighten the constraints to $Cs3$ for the next simulation. Again, the two algorithms give similar trajectory tracking performance as shown in Figs. 5.16(a) and 5.17(a). Both algorithms give the same cost function value (i.e. $J = 5.91 \times 10^{-9}$). Also, the average time consumed per iteration is very similar for both algorithms. However, the number of iterations required to solve the trajectory tracking problem using FL-ASM is again smaller than the number of iterations required using NF-ASM as shown in Table 5.11. The reason again is due to the fact that not all constraints stay active during the simulation as shown in Figs. 5.16(d) and 5.17(d).

For the fourth and fifth simulations, the two algorithms give similar trajectory tracking performance as shown in Figs. 5.19, 5.20, 5.22 and 5.23. Both algorithms give the same cost function value for the fourth and fifth simulation, i.e. $J = 5.046 \times 10^{-9}$ and $J = 5.048 \times 10^{-9}$, respectively. Also, the average time consumed per iteration is very similar for both algorithms. However, the number of iterations required to solve the trajectory tracking problem using NF-ASM is much smaller than the number of iterations required using FL-ASM as shown in Table 5.11. The reason is that the constraints stay active for a longer period during these simulations as compared to the previous cases.

The results, which are summarized in Table 5.11, show that NF-ASM gives better performance in terms of number of iterations when the constraints are active for a longer period during the simulation. FL-ASM is a better choice if the constraint are active only for a fraction of the total time of the simulation as in the case when $Cs2$ and $Cs3$ are used.

Constraints become active when the magnitude of the inputs required to achieve the desired set-points is more than the limits imposed on the inputs. The inputs in this case are the linear velocity error and the steering velocity error. Therefore, the constraints on the inputs will become active when the linear and steering velocities change by a magnitude that exceeds the limit imposed by the constraints. From a practical perspective, it means that the constraints have a higher chance of staying active for a longer period during the trajectory if the speed and the steering velocity keep on changing. Hence, if there are a lot of turns and a lot of obstacles (or traffic) around the WMR (or a vehicle), NF-ASM would be a better option for trajectory tracking as it gives better results in simulation sets in which the constraints stay active for a longer period (i.e. $Cs4$ and $Cs5$).

5.2.6.2 NF-ASM vs MATLAB's MATLAB's QP algorithm

Table 5.12 reiterates the trajectory tracking results of NF-ASM and MATLAB's QP algorithm for comparison purposes.

Table 5.11: NF-ASM vs FL-ASM for Square Trajectory Tracking

Algorithm	Constraints				Total Cost ($J \times 10^{-9}$)	Number of Iterations
	lower bounds		upper bounds			
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)		
NF-ASM	-1	-1	1	1	34.7	0
FL-ASM	-1	-1	1	1	34.7	0
NF-ASM	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}	15.7	421
FL-ASM	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}	15.7	278
NF-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	5.91	11644
FL-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	5.91	7482
NF-ASM	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}	5.046	21351
FL-ASM	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}	5.046	29939
NF-ASM	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}	5.048	21410
FL-ASM	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}	5.048	31635

When the constraint set $Cs1$ is applied, both NF-ASM and MATLAB's Quadratic Programming algorithm demonstrate similar results in terms of trajectory tracking performance as shown in Figs. 5.10(a) and 5.12(a). Both algorithms give the same cost function value (i.e. $J = 3.47 \times 10^{-8}$) which shows that both algorithms are finding the set of solutions for the trajectory tracking problem. The average time taken per sample is high in the beginning iterations for MATLAB's Quadratic Programming algorithm as seen in Fig. 5.12(b). The total time taken by MATLAB's QP algorithm to solve the problem in this case is about twice as much as the time taken by NF-ASM as shown in Table 5.12.

We tighten the constraints to $Cs2$ to allow them to be active. Again, the two algorithms give similar trajectory tracking performance as shown in Figs. 5.13(a) and 5.15(a). Again, both algorithms give the same cost function value (i.e. $J = 1.57 \times 10^{-8}$) which shows that both algorithms are finding the set of solutions for the trajectory tracking problem. However, the total time taken by MATLAB's QP algorithm to solve the problem is again twice as much as the time taken by NF-ASM.

We further tighten the constraints to Cs3 for the next simulation. Again, the two algorithms

Table 5.12: NF-ASM vs MATLAB's QP algorithm for Square Trajectory Tracking

Algorithm	Constraints				Total Cost ($J \times 10^{-9}$)	Total Time Consumed
	lower bounds		upper bounds			
	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)	$v_1 - v_{d1}$ (m/s)	$v_2 - v_{d2}$ (rad/s)		
NF-ASM	-1	-1	1	1	34.7	8.47
MATLAB's QP	-1	-1	1	1	34.7	17.8
NF-ASM	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}	15.7	8.5
MATLAB's QP	-10^{-4}	-10^{-4}	10^{-4}	10^{-4}	15.7	17.8
NF-ASM	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	5.91	11.7
MATLAB's QP	-10^{-5}	-10^{-5}	10^{-5}	10^{-5}	5.91	25.4
NF-ASM	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}	5.046	18
MATLAB's QP	-10^{-8}	-10^{-8}	10^{-8}	10^{-8}	5.046	40
NF-ASM	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}	5.048	18
MATLAB's QP	-10^{-9}	-10^{-9}	10^{-9}	10^{-9}	5.048	40.2

give similar trajectory tracking performance as shown in Figs. 5.16(a) and 5.18(a). Both algorithms give the same cost function value $J = 5.91 \times 10^{-9}$. However, the total time taken by MATLAB's QP algorithm to solve the problem is again twice as much as the time taken by NF-ASM. Similarly, both algorithm show same trajectory tracking performance under constraint sets $Cs4$ and $Cs5$. But again, the time consumed by MATLAB's QP algorithm to solve the tracking problem is twice as much as the time taken by NF-ASM.

The results, which are summarized in Table 5.12, show that NF-ASM gives better performance in terms of total time consumed in all five simulation cases.

Chapter 6

Conclusions

In this research, we explore different control techniques that can be used to perform trajectory tracking of a nonholonomic WMR. We focus on the application of Model Predictive Control (MPC) for trajectory tracking of the WMR because it allows for an easy application of input and state constraints that may arise in practical situations. For example, a designer may want to bound the speed of the WMR or bound the area of movement of the WMR. By using MPC, the designer can easily accomplish this using concrete mathematical equations which guarantee the stability of the control system. MPC also allows easy tuning of the final control law to improve the transient performance and also allows for easy control design optimization. The design optimization can be done to ensure that the error between the actual and desired trajectory is minimized. Also, one can minimize the control effort required to solve the trajectory tracking problem using the built-in optimization method in MPC.

Computation time is the biggest hurdle in adapting MPC strategies for trajectory tracking. This research applies a non-feasible active set MPC algorithm (also known as NF-ASM) that is much faster than the traditional active set methods. It is shown in [1] that

NF-ASM is faster than other traditional algorithms when applied to plant control applications. These plant models, however, are time-invariant. This research, on the other hand, applies NF-ASM to time-varying model of a WMR.

A discrete-time linear time-varying model of a general WMR is used for the simulation. The two inputs to this linear model are the difference in the actual and desired linear velocities $(v_1 - v_{d1})$ and the difference in the actual and desired steering velocities $(v_2 - v_{d2})$. In this research, we put constraints on these control inputs. In other words, we put constraints on the deviation of the linear and steering velocities from their reference values. The control optimization is done to minimize the control effort (i.e. to minimize the magnitude of control inputs required to solve the tracking problem) and the tracking error. MPC's inherent ability to incorporate input constraints and perform control optimization makes this possible in a very simple and systematic manner.

The non-holonomy of the WMR, i.e. its inability to perform certain maneuvers in an instantaneous manner, puts a limitation on the type of reference trajectory that can be tracked. To ensure that the trajectory meets the nonholonomic conditions, we generate reference trajectories using the differential equations of the WMR model. Circular and square trajectories are generated using the differential equations model of the WMR. The vertices of the square trajectory are replaced by parts of a circle to ensure that the trajectory is smooth to track (i.e. no instantaneous right-angled turns are required).

We linearize and discretize the WMR model and perform MATLAB simulations to track these circular and square trajectories. The performance of NF-ASM is compared with the performance of a traditional active set method known as the Fletcher's active set method (FL-ASM) and the QP algorithm commercially available with the MATLAB software.

It is also shown that, although all the three algorithms are capable of providing satisfactory real-time trajectory tracking performance, NF-ASM is a better choice in terms of the simulation time and required number of iterations for trajectory tracking if the constraints on the inputs stay active for a long period during the simulation. On the other hand, FL-ASM proves to be a better option for trajectory tracking if the constraints on the inputs of the WMR model do not stay active for a long period during the trajectory tracking process. MATLAB's QP algorithm takes the most amount of time out of the three algorithms to complete the square as well as the circular trajectory tracking.

Constraints become active when the magnitude of the inputs required to achieve the desired set-point is greater than the magnitude of the limits imposed on the inputs. The inputs in this case are the linear velocity error and the steering velocity error. Therefore, the constraints on the inputs will become active when the linear and steering velocities change by a magnitude that exceeds the limit imposed by the constraints. From a practical perspective, it means that the constraints have a higher chance of staying active for a longer period during the trajectory if the speed and the steering velocity keep on changing. Hence, if there are a lot of turns and a lot of obstacles (or traffic) around the WMR (or a vehicle), NF-ASM would be a better option for trajectory tracking as it gives better results with the simulation sets in which the constraints stay active for a longer period.

Chapter 7

Appendices

7.1 MATLAB Code for Circular Reference Trajectory Generation

Defining the Differential Equations Model:

```
function [F] = car_xdot( t, x )

%% Paramters for Circular Motion
R = 1;
w = 1;
L = 1;
vd1 = R*w;
vd2 = 0;

%% Diff Eq
F(1,1) = vd1*cos(x(3));
F(2,1) = vd1*sin(x(3));
F(3,1) = vd1*tan(x(4))/L;
F(4,1) = vd2;
end
```

ODE45 to Solve Differential Equations:

```
Tinit = 0;
Tfinal = 6.3;
Tvec=[Tinit, Tfinal];
```

```
X0 = [0,0,0,atan(1)];
[Tout, Xout] = ode45( @car_xdot, Tvec, X0)

% plotting your results
figure(1)
plot(Tout, Xout);
plot(Tout, Xout, 'linewidth',1.5)
title('Time response of the system','fontsize',16)
xlabel('time (s)')
ylabel('magnitude of states')
figure(2)
plot(Xout(:,1),Xout(:,2))
xlabel('x','fontsize',14)
ylabel('y','fontsize',14)
title('Plot of x vs y - Ref. Trajectory')
```

7.2 MATLAB Code for Square Reference Trajectory Generation

Defining the Differential Equations Model:

```
function [F] = car_xdot( t, x )

%% Paramters for Square Trajectory Generation
L = 1;
vd1 = 1;
vd2 = 0;

%% Diff Eq
F(1,1) = vd1*cos(x(3));
F(2,1) = vd1*sin(x(3));
F(3,1) = vd1*tan(x(4))/L;
F(4,1) = vd2;
end
```

ODE45 to Solve Differential Equations:

```
Tinit = 0;
Tfinal = 1;
Tvec=[Tinit, Tfinal];

% Vert. Up
X01 = [0,0,pi/2,0];
[Tout1, Xout1] = ode45( @car_xdot, Tvec, X01);

% Hor. Right
X02 = [0,0,0,0];
[Tout2, Xout2] = ode45( @car_xdot, Tvec, X02);

%% Square Trajectory
[m1,n1] = size(Xout1);
[m2,n2] = size(Xout2);

% Vert up
Xout = Xout1;
```

```
% The turn
Tinit = 0;
Tfinal = .83;
Tvec=[Tinit, Tfinal];
X03 = [0,0,pi/2,-pi/180*60];
[Tout3, Xout3] = ode45( @car_xdot, Tvec, X03)
Xout(m1+1:m1+m2,:) = Xout3;
Xout(m1+1:m1+m2,2) = Xout(m1+1:m1+m2,2)+1;

% Hor right
[m,n] = size(Xout);
Xout(m+1:m+m2,:) = Xout2;
Xout(m+1:m+m2,2) = Xout(m+1:m+m2,2)+1.572;
Xout(m+1:m+m2,1) = Xout(m+1:m+m2,1)+0.5007;

% The turn
[m,n] = size(Xout);
Tinit = 0;
Tfinal = .83;
Tvec=[Tinit, Tfinal];
X04 = [0,0,0,-pi/180*60];
[Tout4, Xout4] = ode45( @car_xdot, Tvec, X04)
Xout(m+1:m+m2,:) = Xout4;
Xout(m+1:m+m2,2) = Xout(m+1:m+m2,2)+1.572;
Xout(m+1:m+m2,1) = Xout(m+1:m+m2,1)+1.501;

% Vert dn
[m,n] = size(Xout);
Xout(m+1:m+m1,:) = flipud(Xout1);
Xout(m+1:m+m1,1)= Xout(m+1:m+m1,1)+2.075;

% The turn
[m,n] = size(Xout);
Tinit = 0;
Tfinal = 0.83;
Tvec=[Tinit, Tfinal];
X05 = [0,0,-pi/2,-pi/180*60];
[Tout5, Xout5] = ode45( @car_xdot, Tvec, X05)
Xout(m+1:m+m2,:) = Xout5;
Xout(m+1:m+m2,2) = Xout(m+1:m+m2,2);
```

```
Xout(m+1:m+m2,1) = Xout(m+1:m+m2,1)+2.075;

% Hor Left
[m,n] = size(Xout);
Tinit = 0;
Tfinal = 1;
Tvec=[Tinit, Tfinal];
Xout(m+1:m+m2,:) = flipud(Xout2)+0.574;
Xout(m+1:m+m2,2)= Xout(m+1:m+m2,2)-(2*0.574);

% The final turn
[m,n] = size(Xout);
Tinit = 0;
Tfinal = 0.9;
Tvec=[Tinit, Tfinal];
X06 = [0,0,-pi,-pi/180*60];
[Tout6, Xout6] = ode45( @car_xdot, Tvec, X06)
Xout(m+1:m+m2,:) = Xout6;
Xout(m+1:m+m2,2) = Xout(m+1:m+m2,2)-0.574;
Xout(m+1:m+m2,1) = Xout(m+1:m+m2,1)+0.5748;

%% Plotting Your Results
figure(1)
plot(Xout(:,1),Xout(:,2), 'linewidth',1)
xlabel('x','fontsize',14)
ylabel('y','fontsize',14)
title('Plot of x vs y - Ref. Trajectory','fontsize',14)
AXIS ([-1 3 -1.5 2.5])
```

7.3 MATLAB Code for Trajectory Tracking Using NF-ASM

```
%% Tracking a non constant reference trajectory
Nx=4;
Nu = 2;
X0 = [0.0001;0.0001;0.0001;0.0001];
U0 = zeros(Nu,1);
Yd = zeros(4,1);
Y_total = [0.0001 0.0001 0.0001 0.0001];
U_total = [0 0];
[Nr,Nc] = size(Xout); % Nr is the number of rows of Xout
TimePerIter_c1=[];
NumOfIter_c1=[];
NumOfActCons_c1=[];
NumOfActCons_f1=[];
NumOfIter_f1=[];
TimePerIter_f1=[];

% Mobile Robot Parameters
c = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
L = 1;
Rr = 1;
w = 1;

% Mobile Robot variable Model
vd1 = Rr*w; % For circular trajectory
vd2 = 0;
for i= 1:1:Nr
t_d = Xout(i, 3);
phi_d = Xout(i,4);
a = [0 0 -vd1*sin(t_d) 0;0 0 vd1*cos(t_d) 0;
     0 0 0 (vd1/L)*(cos(phi_d))^2;0 0 0 0];
b = [cos(t_d) 0;sin(t_d) 0;(1/L)*tan(phi_d) 0;0 1];
[ryt,cyt] = size(Y_total);
[rut,cut] = size(U_total);

% Driving the error to zero
Yd(1) = 0;
Yd(2) = 0;
```

```
Yd(3) = 0;
Yd(4) = 0;
Tsim = 3;

% Running the MPC code for NF-ASM
RUN_SIMULATION_MPC

% NF-ASM Parameters
Y_total(ryt+1:(ryt+1)+Tsim,:) = Y_qprog_c1;
U_total(rut+1:(rut+1)+Tsim,:) = U_qprog_c1;
[NrT,NcT]=size(TimePerIter_c1);
TimePerIter_c1(1,NcT+1:(NcT)+Tsim)= timer.iter.qprog_c1;
[NrN,NcN]=size(NumOfIter_c1);
NumOfIter_c1(1,NcT+1:(NcT)+Tsim) = Var_qprog_c1.k;
[NrA,NcA] = size(NumOfActCons_c1);
NumOfActCons_c1(1,NcA+1:(NcA)+Tsim) = Var_qprog_c1.Nb;

% Renewing Initial Conditions
X0 = Xout(i+1,:)-Xout(i,:);
X0=X0';
end
[ryt,cyt] = size(Y_total);
for i= 1:1:Nr
   Xout_temp((i-1)*Tsim+i:i*Tsim+i,:)
   = repmat(Xout(i,:),Tsim+1,1);
end
Y_total(2:ryt,:)= Y_total(2:ryt,:)+Xout_temp;
[ryt,cyt] = size(Y_total);
[rut,cut] = size(U_total);
figure(4)
subplot(2,2,1)
plot(Y_total(:,1),Y_total(:,2))
title('(a) Position of the WMR')
xlabel('x-axis (meters)')
ylabel ('y-axis (meters)')
AXIS ([-1 3 -1.5 2.5]) % For Square Trajectory
%AXIS ([-1.1 1.1 -.1 2.1]) % For Circular Trajectory

% Plots related to NF-ASM
subplot(2,2,2)
```

```
plot(TimePerIter_c1)
title('(b) Time per Iteration')
xlabel('Time Sample (seconds)')
ylabel ('Time Consumed (seconds)')
AXIS ([0 max(size(TimePerIter_c1))+
.025*max(size(TimePerIter_c1)) min(TimePerIter_c1)
max(TimePerIter_c1)+.05*max(TimePerIter_c1)]);
subplot(2,2,3)
plot(NumOfIter_c1)
title('(c) Number of Iterations Used')
xlabel('Time sample (seconds)')
ylabel ('Number of Iterations')
AXIS ([0 max(size(NumOfIter_c1))
+.025*max(size(NumOfIter_c1))
min(NumOfIter_c1) max(NumOfIter_c1)
+.05*max(NumOfIter_c1)]);
%AXIS ([0 max(size(NumOfIter_c1))
+.025*max(size(NumOfIter_c1))
-1 1]); % For unconstrained solution
subplot(2,2,4)
plot(NumOfActCons_c1)
title('(d) Number of Active Constraints')
xlabel('Time sample (seconds)')
ylabel ('Number of Active Constraints')
AXIS ([0 max(size(NumOfActCons_c1))+
.025*max(size(NumOfActCons_c1)) 0
max(NumOfActCons_c1)+
.05*max(NumOfActCons_c1)]);
%AXIS ([0 max(size(NumOfActCons_c1))
+.025*max(size(NumOfActCons_c1))
-1 1]); % For unconstrained solution
saveas(gcf, 'p1_c1_s7', 'eps')
figure (5)
subplot(1,2,1)
semilogx(1:1:size(Y_total(2:ryt,3)),
Y_total(2:ryt,3)-Xout_temp(:,3))
title({'(a) Orientation Error';
'(theta-theta_d)'})
xlabel('Time sample (s)')
ylabel ('Orientation Error Magnitude (radians)')
```

```
AXIS ([0 max(size(Y_total(2:ryt,3)))
+.025*max(size(Y_total(2:ryt,3)))
min(Y_total(2:ryt,3)-Xout_temp(:,3))
max(Y_total(2:ryt,3)
-Xout_temp(:,3))+.5*max(Y_total(2:ryt,3)
-Xout_temp(:,3))]);
subplot(1,2,2, 'XScale', 'Log')
semilogx(1:1:size(Y_total(2:ryt,4)),
Y_total(2:ryt,4)-Xout_temp(:,4))
title({'(b) Steering Error'; '(phi-phi_d)'})
xlabel('Time sample (seconds)')
ylabel ('Steering Angle Error Magnitude (radians)')
AXIS ([0 max(size(Y_total(2:ryt,4)))
+.025*max(size(Y_total(2:ryt,4)))
min(Y_total(2:ryt,4)
-Xout_temp(:,4)) max(Y_total(2:ryt,4)
-Xout_temp(:,4))+.9*max(Y_total(2:ryt,4)
-Xout_temp(:,4))]);
saveas(gcf, 'p2_c1_s7', 'eps')
figure(6)
subplot(1,2,1)
semilogx(Y_total(2:ryt,1)-Xout_temp(:,1))
title({'(a) X-Position Error'; '(x-x_d)'})
xlabel('Time sample (seconds)')
ylabel ('X-coordinate Position Error Magnitude (meters)')
AXIS ([0 max(size(Y_total(2:ryt,1)))
-.025*max(size(Y_total(2:ryt,1))) min(Y_total(2:ryt,1)
-Xout_temp(:,1)) max(Y_total(2:ryt,1)-Xout_temp(:,1))
+.9*max(Y_total(2:ryt,1)-Xout_temp(:,1))]);
subplot(1,2,2)
semilogx(Y_total(2:ryt,2)-Xout_temp(:,2))
title({'(b) Y-Position Error'; '(y-y_d)'})
xlabel('Time sample (seconds)')
ylabel ('Y-coordinate Position Error Magnitude (meters)')
AXIS ([0 max(size(Y_total(2:ryt,4)))
+.025*max(size(Y_total(2:ryt,4)))
min(Y_total(2:ryt,2)-Xout_temp(:,2)) max(Y_total(2:ryt,2)
-Xout_temp(:,2))+.9*max(Y_total(2:ryt,2)
-Xout_temp(:,2))]);
saveas(gcf, 'p3_c1_s7', 'eps')
```

```
figure(7)
subplot(1,2,1)
semilogx(U_total(:,1))
title({'(a) Linear Velocity Input'; '(v_1 - v_d_1)'})
xlabel('Time sample (seconds)')
ylabel ('Linear Velocity Input Magnitude (meters/second)')
AXIS ([0 max(size(U_total(2:ryt,1)))
+.025*max(size(U_total(2:ryt,1)))
lb(1)+.25*lb(1)
ub(1)+.25*ub(1)]); % Not for unconstrained solution
subplot(1,2,2)
semilogx(U_total(:,2))
title({'(b) Steering Velocity Input'; '(v_2 - v_d_2)'})
xlabel('Time sample (seconds)')
ylabel ('Steering Velocity Input Magnitude(radians/second)')
AXIS ([0 max(size(U_total(2:ryt,2)))
+.025*max(size(U_total(2:ryt,2)))
lb(2)+.25*lb(2) ub(2)
+.25*ub(2)]); % Not for unconstrained solution
saveas(gcf, 'p4_c1_s7', 'eps')
sprintf('Num of Iter c1= %d', sum(NumOfIter_c1))
sprintf('Time per Iter c1= %d', sum(TimePerIter_c1))
sprintf('Cost of c1= %d', Jval_my)
```

7.4 MATLAB Code for Trajectory Tracking Using FL-ASM

```
%% Tracking a non constant reference trajectory
Nx=4;
Nu = 2;
X0 = [0.0001;0.0001;0.0001;0.0001];
U0 = zeros(Nu,1);
Yd = zeros(4,1);
Y_total = [0.0001 0.0001 0.0001 0.0001];
U_total = [0 0];
[Nr,Nc] = size(Xout); % Nr is the number of rows of Xout
TimePerIter_c1=[];
NumOfIter_c1=[];
NumOfActCons_c1=[];
NumOfActCons_f1=[];
NumOfIter_f1=[];
TimePerIter_f1=[];

% Mobile Robot Parameters
c = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
L = 1;
Rr = 1;
w = 1;

% Mobile Robot variable Model
vd1 = Rr*w; % For circular trajectory
vd2 = 0;
for i= 1:1:Nr
t_d = Xout(i, 3);
phi_d = Xout(i,4);
a = [0 0 -vd1*sin(t_d) 0;0 0 vd1*cos(t_d) 0;0 0 0
(vd1/L)*(cos(phi_d))^2;0 0 0 0];
b = [cos(t_d) 0;sin(t_d) 0;(1/L)*tan(phi_d) 0;0 1];
[ryt,cyt] = size(Y_total);
[rut,cut] = size(U_total);

% Driving the error to zero
Yd(1) = 0;
Yd(2) = 0;
```

```
Yd(3) = 0;
Yd(4) = 0;
Tsim = 3;

% Running the MPC Code for FL-ASM
RUN_SIMULATION_MPC

% Fletcher's ASM Parameters
Y_total(ryt+1:(ryt+1)+Tsim,:) = Y_qprog_f1;
U_total(rut+1:(rut+1)+Tsim,:) = U_qprog_f1;
[NrT,NcT]=size(TimePerIter_f1);
TimePerIter_f1(1,NcT+1:(NcT)+Tsim)= timer.iter.qprog_f1;
[NrN,NcN]=size(NumOfIter_f1);
NumOfIter_f1(1,NcT+1:(NcT)+Tsim) = Var_qprog_f1.k;
[NrA,NcA] = size(NumOfActCons_f1);
NumOfActCons_f1(1,NcA+1:(NcA)+Tsim) = Var_qprog_f1.Nb;

% Renewing Initial Conditions
X0 = Xout(i+1,:)-Xout(i,:);
X0=X0';
end
[ryt,cyt] = size(Y_total);
for i= 1:1:Nr
Xout_temp((i-1)*Tsim+i:i*Tsim+i,:)
= repmat(Xout(i,:),Tsim+1,1);
end
Y_total(2:ryt,:)= Y_total(2:ryt,:)+Xout_temp;
[ryt,cyt] = size(Y_total);
[rut,cut] = size(U_total);
figure(4)
subplot(2,2,1)
plot(Y_total(:,1),Y_total(:,2))
title('(a) Position of the WMR')
xlabel('x-axis (meters)')
ylabel ('y-axis (meters)')
AXIS ([-1 3 -1.5 2.5]) % For Square Trajectory
%AXIS ([-1.1 1.1 -.1 2.1]) % For Circular Trajectory

% Plots related to Fletcher's Algorithm
subplot(2,2,2)
```

```
plot(TimePerIter_f1)
title('(b) Time per Iteration')
xlabel('Iteration Number')
ylabel ('Time Consumed (seconds)')
AXIS ([0 max(size(TimePerIter_f1))
+.025*max(size(TimePerIter_f1))
min(TimePerIter_f1) max(TimePerIter_f1)
+.05*max(TimePerIter_f1)]);
subplot(2,2,3)
plot(NumOfIter_f1)
title('(c) Number of Iterations Used')
xlabel('Time sample (s)')
ylabel ('Number of Iterations')
AXIS ([0 max(size(NumOfIter_f1))
+.025*max(size(NumOfIter_f1))
min(NumOfIter_f1) max(NumOfIter_f1)
+.05*max(NumOfIter_f1)]);
%AXIS ([0 max(size(NumOfIter_f1))
+.025*max(size(NumOfIter_f1))
        -1 1]); % For unconstrained solution
subplot(2,2,4)
plot(NumOfActCons_f1)
title('(d) Number of Active Constraints')
xlabel('Time sample (s)')
ylabel ('Number of Active Constraints')
AXIS ([0 max(size(NumOfActCons_f1))
+.025*max(size(NumOfActCons_f1))
0 max(NumOfActCons_f1)+.05*max(NumOfActCons_f1)]);
%AXIS ([0 max(size(NumOfActCons_f1))
+.025*max(size(NumOfActCons_f1))
-1 1]); % For unconstrained solution
saveas(gcf, 'p1_f1_s7', 'eps')
figure (5)
subplot(1,2,1)
semilogx(1:1:size(Y_total(2:ryt,3)),Y_total(2:ryt,3)
-Xout_temp(:,3))
title({'(a) Orientation Error'; '(theta-theta_d)'})
xlabel('Time sample (s)')
ylabel ('Orientation Error Magnitude (radians)')
AXIS ([0 max(size(Y_total(2:ryt,3)))
```

```
+.025*max(size(Y_total(2:ryt,3)))
min(Y_total(2:ryt,3)-Xout_temp(:,3)) max(Y_total(2:ryt,3)
-Xout_temp(:,3))+.5*max(Y_total(2:ryt,3)-Xout_temp(:,3))]);
subplot(1,2,2, 'XScale', 'Log')
semilogx(1:1:size(Y_total(2:ryt,4)),Y_total(2:ryt,4)
-Xout_temp(:,4))
title({'(b) Steering Error'; '(phi-phi_d)'})
xlabel('Time sample (seconds)')
ylabel ('Steering Angle Error Magnitude (radians)')
AXIS ([0 max(size(Y_total(2:ryt,4)))
+.025*max(size(Y_total(2:ryt,4)))
min(Y_total(2:ryt,4)-Xout_temp(:,4)) max(Y_total(2:ryt,4)
-Xout_temp(:,4))+.9*max(Y_total(2:ryt,4)-Xout_temp(:,4))]);
saveas(gcf, 'p2_f1_s7', 'eps')
figure(6)
subplot(1,2,1)
semilogx(Y_total(2:ryt,1)-Xout_temp(:,1))
title({'(a) X-Position Error'; '(x-x_d)'})
xlabel('Time sample (seconds)')
ylabel ('X-coordinate Position Error Magnitude (meters)')
AXIS ([0 max(size(Y_total(2:ryt,1)))
-.025*max(size(Y_total(2:ryt,1)))
min(Y_total(2:ryt,1)-Xout_temp(:,1)) max(Y_total(2:ryt,1)
-Xout_temp(:,1))+.9*max(Y_total(2:ryt,1)-Xout_temp(:,1))]);
subplot(1,2,2)
semilogx(Y_total(2:ryt,2)-Xout_temp(:,2))
title({'(b) Y-Position Error'; '(y-y_d)'})
xlabel('Time sample (seconds)')
ylabel ('Y-coordinate Position Error Magnitude (meters)')
AXIS ([0 max(size(Y_total(2:ryt,4)))
+.025*max(size(Y_total(2:ryt,4)))
min(Y_total(2:ryt,2)-Xout_temp(:,2))max(Y_total(2:ryt,2)
-Xout_temp(:,2))+.9*max(Y_total(2:ryt,2)-Xout_temp(:,2))]);
saveas(gcf, 'p3_f1_s7', 'eps')
figure(7)
subplot(1,2,1)
semilogx(U_total(:,1))
title({'(a) Linear Velocity Input'; '(v_1 - v_d_1)'})
xlabel('Time sample (seconds)')
ylabel ('Linear Velocity Input Magnitude (meters/second)')
```

```
AXIS ([0 max(size(U_total(2:ryt,1)))
+.025*max(size(U_total(2:ryt,1))) lb(1)+.25*lb(1)
ub(1)+.25*ub(1)]); % Not for unconstrained solution
subplot(1,2,2)
semilogx(U_total(:,2))
title({'(b) Steering Velocity Input'; '(v_2 - v_d_2)'})
xlabel('Time sample (seconds)')
ylabel ('Steering Velocity Input Magnitude (radians/second)')
AXIS ([0 max(size(U_total(2:ryt,2)))
+.025*max(size(U_total(2:ryt,2)))lb(2)+.25*lb(2)
ub(2)+.25*ub(2)]); % Not for unconstrained solution
saveas(gcf, 'p4_f1_s7', 'eps')
sprintf('num of iter f1= %d', sum(NumOfIter_f1))
sprintf('total time f1= %d', sum(TimePerIter_f1))
sprintf('cost of f1= %d', Jval_f1)
```

7.5 MATLAB Code for Trajectory Tracking Using MATLAB's QP

```
%% Tracking a non constant reference trajectory
Nx=4;
Nu = 2;
X0 = [0.0001;0.0001;0.0001;0.0001];
U0 = zeros(Nu,1);
Yd = zeros(4,1);
Y_total = [0.0001 0.0001 0.0001 0.0001];
U_total = [0 0];
[Nr,Nc] = size(Xout); % Nr is the number of rows of Xout
TimePerIter_c1=[];
NumOfIter_c1=[];
NumOfActCons_c1=[];
NumOfActCons_f1=[];
NumOfIter_f1=[];
TimePerIter_f1=[];

% Mobile Robot Parameters
c = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
L = 1;
Rr = 1;
w = 1;

% Mobile Robot variable Model
vd1 = Rr*w; % For circular trajectory
vd2 = 0;
for i= 1:1:Nr
t_d = Xout(i, 3);
phi_d = Xout(i,4);
a = [0 0 -vd1*sin(t_d) 0;0 0 vd1*cos(t_d) 0;0 0 0
     (vd1/L)*(cos(phi_d))^2;0 0 0 0];
b = [cos(t_d) 0;sin(t_d) 0;(1/L)*tan(phi_d) 0;0 1];
[ryt,cyt] = size(Y_total);
[rut,cut] = size(U_total);

% Driving the error to zero
Yd(1) = 0;
Yd(2) = 0;
```

```
Yd(3) = 0;
Yd(4) = 0;
Tsim = 3;

% Running the MPC code for MATLAB's Quadratic
Programming algorithm
RUN_SIMULATION_MPC

% MATLAB's Quadratic Programming algorithm Parameters
Y_total(ryt+1:(ryt+1)+Tsim,:) = Y_qp;
U_total(rut+1:(rut+1)+Tsim,:) = U_qp;
[NrT,NcT]=size(TimePerIter_c1);
TimePerIter_c1(1,NcT+1:(NcT)+Tsim)= timer.iter.qp;
[NrN,NcN]=size(NumOfIter_c1);
%NumOfIter_c1(1,NcT+1:(NcT)+Tsim) = Var_qp.k;
[NrA,NcA] = size(NumOfActCons_c1);
%NumOfActCons_c1(1,NcA+1:(NcA)+Tsim) = Var_qp.Nb;

% Renewing Initial Conditions
X0 = Xout(i+1,:)-Xout(i,:);
X0=X0';
end
[ryt,cyt] = size(Y_total);
for i= 1:1:Nr
   Xout_temp((i-1)*Tsim+i:i*Tsim+i,:)
   = repmat(Xout(i,:),Tsim+1,1);
end
Y_total(2:ryt,:)= Y_total(2:ryt,:)+Xout_temp;
[ryt,cyt] = size(Y_total);
[rut,cut] = size(U_total);
figure(4)
subplot(1,2,1)
plot(Y_total(:,1),Y_total(:,2))
title('(a) Position of the WMR')
xlabel('x-axis (meters)')
ylabel ('y-axis (meters)')
AXIS ([-1 3 -1.5 2.5]) % For Square Trajectory
%AXIS ([-1.1 1.1 -.1 2.1]) % For Circular Trajectory
subplot(1,2,2)
plot(TimePerIter_c1)
```

```
title('(b) Time per Iteration')
xlabel('Time Sample (seconds)')
ylabel ('Time Consumed (seconds)')
AXIS ([0 max(size(TimePerIter_c1))
+.025*max(size(TimePerIter_c1))
min(TimePerIter_c1) max(TimePerIter_c1)
+.05*max(TimePerIter_c1)]);
saveas(gcf, 'p1_qp_s7', 'eps')
figure (5)
subplot(1,2,1)
semilogx(1:1:size(Y_total(2:ryt,3)),Y_total(2:ryt,3)
-Xout_temp(:,3))
title({'(a) Orientation Error'; '(theta-theta_d)'})
xlabel('Time sample (s)')
ylabel ('Orientation Error Magnitude (radians)')
AXIS ([0 max(size(Y_total(2:ryt,3)))
+.025*max(size(Y_total(2:ryt,3)))
min(Y_total(2:ryt,3)-Xout_temp(:,3)) max(Y_total(2:ryt,3)
-Xout_temp(:,3))+.5*max(Y_total(2:ryt,3)-Xout_temp(:,3))]);
subplot(1,2,2, 'XScale', 'Log')
semilogx(1:1:size(Y_total(2:ryt,4)),Y_total(2:ryt,4)
-Xout_temp(:,4))
title({'(b) Steering Error'; '(phi-phi_d)'})
xlabel('Time sample (seconds)')
ylabel ('Steering Angle Error Magnitude (radians)')
AXIS ([0 max(size(Y_total(2:ryt,4)))
+.025*max(size(Y_total(2:ryt,4)))
min(Y_total(2:ryt,4)-Xout_temp(:,4))
max(Y_total(2:ryt,4)-Xout_temp(:,4))
+.9*max(Y_total(2:ryt,4)-Xout_temp(:,4))]);
saveas(gcf, 'p2_qp_s7', 'eps')
figure(6)
subplot(1,2,1)
semilogx(Y_total(2:ryt,1)-Xout_temp(:,1))
title({'(a) X-Position Error'; '(x-x_d)'})
xlabel('Time sample (seconds)')
ylabel ('X-coordinate Position Error Magnitude (meters)')
AXIS ([0 max(size(Y_total(2:ryt,1)))
-.025*max(size(Y_total(2:ryt,1)))
min(Y_total(2:ryt,1)-Xout_temp(:,1))
```

```
max(Y_total(2:ryt,1)-Xout_temp(:,1))
+.9*max(Y_total(2:ryt,1)-Xout_temp(:,1))]);
subplot(1,2,2)
semilogx(Y_total(2:ryt,2)-Xout_temp(:,2))
title({'(b) Y-Position Error'; '(y-y_d)'})
xlabel('Time sample (seconds)')
ylabel ('Y-coordinate Position Error Magnitude (meters)')
AXIS ([0 max(size(Y_total(2:ryt,4)))
+.025*max(size(Y_total(2:ryt,4)))
min(Y_total(2:ryt,2)-Xout_temp(:,2)) max(Y_total(2:ryt,2)
-Xout_temp(:,2))+.9*max(Y_total(2:ryt,2)-Xout_temp(:,2))]);
saveas(gcf, 'p3_qp_s7', 'eps')
figure(7)
subplot(1,2,1)
semilogx(U_total(:,1))
title({'(a) Linear Velocity Input'; '(v_1 - v_d_1)'})
xlabel('Time sample (seconds)')
ylabel ('Linear Velocity Input Magnitude (meters/second)')
AXIS ([0 max(size(U_total(2:ryt,1)))
+.025*max(size(U_total(2:ryt,1))) lb(1)+.25*lb(1)
ub(1)+.25*ub(1)]); % Not for unconstrained solution
subplot(1,2,2)
semilogx(U_total(:,2))
title({'(b) Steering Velocity Input'; '(v_2 - v_d_2)'})
xlabel('Time sample (seconds)')
ylabel ('Steering Velocity Input Magnitude (radians/second)')
AXIS ([0 max(size(U_total(2:ryt,2)))
+.025*max(size(U_total(2:ryt,2))) lb(2)+.25*lb(2)
ub(2)+.25*ub(2)]); % Not for unconstrained solution
saveas(gcf, 'p4_qp_s7', 'eps')
sprintf('Total Time for Qprog= %d', sum(TimePerIter_c1))
sprintf('Cost of Qprog= %d', Jval_qp)
```

Bibliography

[1] R. Milman and E. Davison, "A fast MPC algorithm using nonfeasible active set methods," *Journal of Optimization Theory and Applications*, vol. 139, no. 3, pp. 591–616, 2008.

[2] R. Murray, Z. Li, and S. Sastry, *A mathematical introduction to robotic manipulation*. CRC, 1994.

[3] J. Neimark and N. Fufaev, *Dynamics of nonholonomic systems*. Amer Mathematical Society, 1972.

[4] C. De Wit, H. Khennouf, C. Samson, and O. Sordalen, "Nonlinear control design for mobile robots," *Recent trends in mobile robots*, pp. 121–156, 1993.

[5] D. Tilbury and M. Tilbury, "Exterior differential systems and nonholonomic motion planning," *Memo. No. UCB/ERL M94/90, UC Berkeley*, 1994.

[6] J. Laumond, P. Jacobs, M. Taix, and R. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 577–593, 1994.

[7] C. Samson, "Velocity and torque feedback control of a nonholonomic cart," *Advanced robot control*, pp. 125–151, 1991.

[8] ——, "Time-varying feedback stabilization of car-like wheeled mobile robots," *The International journal of robotics research*, vol. 12, no. 1, p. 55, 1993.

[9] C. Samson and K. Ait-Abderrahim, "Feedback stabilization of a nonholonomic wheeled mobile robot," in *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, 1991, pp. 1242–1247.

[10] J. Barraquand and J. Latombe, "On nonholonomic mobile robots and optimal maneuvering," in *IEEE International Symposium on Intelligent Control*, 1989, pp. 340–347.

[11] A. Ollero and O. Amidi, "Predictive path tracking of mobile robots. Application to the CMUNavLab," in *Fifth International Conference on Advanced Robotics*, 1991, pp. 1081–1086.

[12] J. Barraquand and J. Latombe, "Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles," *Algorithmica*, vol. 10, no. 2, pp. 121–155, 1993.

[13] J. Laumond, S. Sekhavat, M. Vaisset *et al.*, "Collision-free motion planning for a nonholonomic mobile robot with trailers," in *4th IFAC Symp. on Robot Control*, pp. 171–177.

[14] C. De Wit and O. Sordalen, "Exponential stabilization of mobile robots with nonholonomic constraints," *IEEE Transactions on Automatic Control*, vol. 37, no. 11, pp. 1791–1797, 1992.

[15] H. Essen and H. Nijmeijer, "Non-linear model predictive control of constrained mobile robots," in *Proc. European Control Conference*, 2001, pp. 1157–1162.

[16] W. Dixon, M. De Queiroz, D. Dawson, and T. Flynn, "Adaptive tracking and regulation of a wheeled mobile robot with controller/update law modularity," *IEEE Transactions on Control Systems Technology*, vol. 12, no. 1, pp. 138–147, 2004.

[17] M. Oya, C. Su, and R. Katoh, "Robust adaptive motion/force tracking control of uncertain nonholonomic mechanical systems," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 175–182, 2003.

[18] O. Sordalen, "Conversion of the kinematics of a car with n trailers into a chained form," in *IEEE International Conference on Robotics and Automation*, 1993, pp. 382–387.

[19] D. Tilbury, R. Murray, and S. Sastry, "Trajectory generation for the n-trailer problem using goursat normal form," in *32nd IEEE Conference on Decision and Control*, 1993, pp. 971–977.

[20] L. Bushnell, D. Tilbury, and S. Sastry, "Steering three-input nonholonomic systems: the fire truck example," *The International Journal of Robotics Research*, vol. 14, no. 4, p. 366, 1995.

[21] A. Sahai, M. Secor, and L. Bushnell, "An obstacle avoidance algorithm for a car pulling many trailers with kingpin hitching," in *33rd IEEE Conf. on Decision and Control*, 1994, pp. 2944–2949.

[22] D. Tilbury, J. Laumond, R. Murray, S. Sastry, and G. Walsh, "Steering car-like systems with trailers using sinusoids," in *IEEE International Conference on Robotics and Automation*, 1992, pp. 1993–1998.

[23] I. Kolmanovsky and N. McClamroch, "Developments in nonholonomic control problems," *IEEE Control systems magazine*, vol. 15, no. 6, pp. 20–36, 1995.

[24] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, 1990.

[25] A. Bloch, M. Reyhanoglu, and N. McClamroch, "Control and stabilization of nonholonomic dynamic systems," *IEEE Transactions on Automatic Control*, vol. 37, no. 11, pp. 1746–1757, 1992.

[26] A. De Luca, G. Oriolo, and C. Samson, "Feedback control of a nonholonomic car-like robot," *Robot motion planning and control*, pp. 171–253, 1998.

[27] F. Kühne, W. Lages, and J. da Silva Jr, "Model predictive control of a mobile robot using linearization," in *Proceedings of Mechatronics and Robotics*, 2004.

[28] R. Fletcher, "Practical Methods of Optimization: Vol. 2: Constrained Optimization." 1981.

[29] G. Walsh, D. Tilbury, S. Sastry, R. Murray, and J. Laumond, "Stabilization of trajectories for systems with nonholonomic constraints," *IEEE Transactions on Automatic Control*, vol. 39, no. 1, pp. 216–222, 1994.

[30] G. Campion, B. d'Andrea Novel, and G. Bastin, "Controllability and state Feedback stabilizability of non holonomic mechanical systems," *Advanced Robot Control*, pp. 106–124.

[31] G. Oriolo and Y. Nakamura, "Control of mechanical systems with second-order nonholonomic constraints: Underactuated manipulators," in *Conference on Decision and Control*. Citeseer, 1991, pp. 2398–2403.

[32] B. Maschke and A. Van der Schaft, "A Hamiltonian approach to stabilization of nonholonomic mechanicalsystems," in *33rd IEEE Conference on Decision and Control*, vol. 3, 1994.

[33] N. Sarkar, X. Yun, and V. Kumar, "Dynamic path following: A new control algorithm for mobile robots," in *32nd IEEE Conference on Decision and Control*, 1993, pp. 2670–2675.

[34] A. Kapitanovsky, A. Goldenberg, and J. Mills, "Dynamic control and motion planning technique for a class of nonlinear systems with drift," *Systems & Control Letters*, vol. 21, no. 5, pp. 363–369, 1993.

[35] N. Getz, "Control of balance for a nonlinear nonholonomic non-minimum phase model of a bicycle," vol. 1, 1994.

[36] C. Rui and N. McClamroch, "Stabilization and asymptotic path tracking of a rolling disk," in *34th IEEE Conference on Decision and Control*, vol. 4, 1995.

[37] J. Latombe, *Robot motion planning*. Springer Verlag, 1990.

[38] Z. Li and J. Canny, *Nonholonomic motion planning*. Kluwer Academic Publishers, 1993.

[39] R. Brockett, "New directions in applied mathematics," *Control Theory and Singular Riemannian Geometry*, pp. 11–27, 1981.

[40] A. Isidori, *Nonlinear control systems*. Springer Verlag, 1995.

[41] A. Bloch and S. Drakunov, "Stabilization of a nonholonomic system via sliding modes," in *33rd IEEE Conference on Decision and Control*, vol. 3, 1994.

[42] C. Samson, "Control of chained systems application to path following and time-varying point-stabilization of mobile robots," *IEEE Transactions on Automatic Control*, vol. 40, no. 1, pp. 64–77, 1995.

[43] J. Hespanha and A. Morse, "Stabilization of nonholonomic integrators via logic-based switching," *University of California, Berkeley*, vol. 94720, 1999.

[44] G. Oriolo, A. De Luca, M. Vendittelli *et al.*, "WMR control via dynamic feedback linearization: design, implementation, and experimental validation," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 6, pp. 835–852, 2002.

[45] O. Sordalen and K. Wichlund, "Exponential stabilization of a car with n trailers," in *32nd IEEE Conference on Decision and Control*, 1993, pp. 978–983.

[46] O. Sordalen and O. Egeland, "Exponential stabilization of nonholonomic chained systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 1, pp. 35–49, 1995.

[47] C. Canudes de Wit, H. Berghuis, and H. Nijmeijer, "Practical stabilization of nonlinear systems in chained form," in *33rd IEEE Conference on Decision and Control*, vol. 4, 1994.

[48] I. Kolmanovsky and N. McClamroch, "Stabilization of nonholonomic Chaplygin systems with linear basespace dynamics," in *34th IEEE Conference on Decision and Control*, vol. 1, 1995.

[49] I. Kolmanovsky, "Discontinuous feedback stabilization of nonholonomic systems in extended power form," in *33rd IEEE Conference on Decision and Control*, vol. 4, 1994, pp. 3469–3474.

[50] I. Kolmanovsky, M. Reyhanoglu, and N. McClamroch, "Switched mode feedback control laws for nonholonomic systems in extended power form," *Systems and Control Letters*, vol. 27, no. 1, pp. 29–36, 1996.

[51] H. Krishnan, M. Reyhanoglu, and H. McClamroch, "Attitude stabilization of a rigid spacecraft using two control torques: A nonlinear control approach based on the spacecraft attitude dynamics," *Automatica*, vol. 30, no. 6, pp. 1023–1027, 1994.

[52] H. Krishnan, N. McClamroch, and M. Reyhanoglu, "Attitude stabilization of a rigid spacecraft using two momentum wheel actuators," in *Efficient Reorientation Maneuvers for Spacecraft with Multiple Articulated Payloads*, 1993.

[53] J. Ortega and E. Camacho, "Mobile robot navigation in a partially structured static environment, using neural predictive control," *Control Engineering Practice*, vol. 4, no. 12, pp. 1669–1679, 1996.

[54] X. Yang, K. He, M. Guo, and B. Zhang, "An intelligent predictive control approach to path tracking problem of autonomous mobile robot," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4, 1998.

[55] G. Welch and G. Bishop, "An introduction to the Kalman filter," *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 1995.

[56] D. Simon and T. Chia, "Kalman filtering with state equality constraints," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 38, no. 1, pp. 128–136, 2002.

[57] F. Lizarralde, E. Nunes, L. Hsu, and J. Wen, "Mobile robot navigation using sensor fusion," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1. IEEE, 2003, pp. 458–463.

[58] J. Wang and W. Wilson, "3D relative position and orientation estimation using Kalman filter for robot control," in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. IEEE, 2002, pp. 2638–2645.

[59] F. Allgöwer, T. Badgwell, J. Qin, J. Rawlings, and S. Wright, "Nonlinear predictive control and moving horizon estimation–An introductory overview," *Highlights of ECC Advances in Control*, vol. 99, pp. 391–449, 1999.

[60] F. Allgöwer, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability," *Automatica Oxford*, vol. 34, no. 10, pp. 1205–1217, 1998.

[61] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica Oxford*, vol. 36, pp. 789–814, 2000.

[62] M. Cannon and B. Kouvaritakis, "Continuous-time predictive control of constrained nonlinear systems," *Nonlinear Model Predictive Control*, vol. 26, pp. 205–215.

[63] J. Guldner and V. Utkin, "Stabilization of non-holonomic mobile robots using Lyapunovfunctions for navigation and sliding mode control," in *33rd IEEE Conference on Decision and Control*, vol. 3, 1994.

[64] R. Milman, "Speedup of the quadratic programming problem and other issues in model predictive control," PhD Dissertation, University of Toronto, Department of Electrical and Computer Engineering, 2004.

[65] M. Lau, S. Yue, K. Ling, and J. Maciejowski, "A comparison of interior point and active set methods for FPGA implementation of model predictive control," in *Proc. European Control Conference*, 2009, pp. 156–161.

[66] J. Normey-Rico, J. Gómez-Ortega, and E. Camacho, "A Smith-predictor-based generalised predictive controller for mobile robot path-tracking," *Control Engineering Practice*, vol. 7, no. 6, pp. 729–740, 1999.

[67] Z. Li, J. Sun, and S. Oh, "Path following for marine surface vessels with rudder and roll constraints: an MPC approach," in *Proceedings of American Control Conference*. IEEE Press, 2009, pp. 3611–3616.

[68] M. Henson, "Nonlinear model predictive control: current status and future directions," *Computers and Chemical Engineering*, vol. 23, no. 2, pp. 187–202, 1998.